9. Trie

Tato část je věnována jedné z nejstarších datových struktur. Její lineární verzí jsou klasické slovníky, ale my se zde seznámíme s její stromovou podobou, která je vhodnější pro počítačové použití. Aby se odlišila od binárních vyhledávacích stromů a (a, b)-stromů, byl pro ni vymyšlen nový název trie. Je to nové slovo odvozené od slova tree – strom, a tím se autoři snažili vystihnout strukturu reprezentace.

Nejprve popíšeme problém. Mějme konečnou abecedu Σ o k písmenech a univerzum U, které je tvořeno všemi slovy nad Σ o délce l. Naším cílem bude reprezentovat množinu $S \subseteq U$ a realizovat operace **MEMBER**, **INSERT** a **DELETE**. Trie je konečný strom takový, že každý vnitřní vrchol má právě k synů a ty jsou v jednoznačné korespondenci s písmeny abecedy. Každému vrcholu v můžeme přiřadit proměnnou slovo(v) definovanou rekurzivně:

> když v je kořen stromu, pak slovo(v) = λ (prázdné slovo);
> když v je a-tý syn vrcholu w pro $a \in \Sigma$, pak slovo(v) = slovo(w)a.

Pak pro každý vnitřní vrchol v trie musí platit, že slovo(v) je prefixem nějakého slova v S. Pro každý list v je definována boolská funkce rep tak, že rep(v) = *true*, právě když slovo(v) $\in S$. Nyní popíšeme jednoduché algoritmy realizující operace **MEMBER**, **INSERT** a **DELETE** v datové struktuře trie.

MEMBER(α)
t :=kořen stromu, $i := 1$
while t není list **do**
  $a := i$-té písmeno α, $t := a$-tý syn t, $i := i + 1$
enddo
if rep(t) = *true* **then** $\alpha \in S$ **else** $\alpha \notin S$ **endif**

INSERT(α)
t :=kořen stromu, $i := 1$
while t není list **do**
  $a := i$-té písmeno α, $t := a$-tý syn t, $i := i + 1$
enddo
while $i \leq l$ **do**
  t změň na vnitřní vrchol stromu
  for every $a \in \Sigma$ **do**
    vytvoř list $t_a = a$-tý syn vrcholu t, rep(t_a) := $false$
  enddo
  $a := i$-té písmeno slova α, $t := a$-tý syn t, $i := i + 1$
enddo
rep(t) :=true

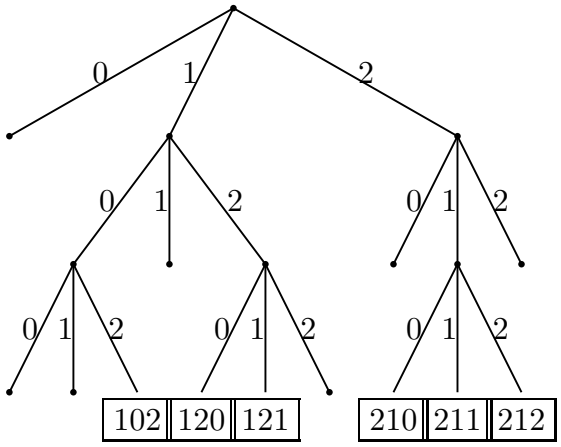
DELETE(α)
t :=kořen stromu, $i := 1$
while t není list **do**
  $a := i$-té písmeno α, $t := a$-tý syn t, $i := i + 1$
enddo

rep(t) := $false$, t := otec(t)
while všichni synové v vrcholu t jsou listy a rep(v) = $false$ **do**
  odstraň všechny syny vrcholu t
  t změň na list, rep(t) := $false$, t := otec(t)
enddo

Protože všechna slova v S mají délku l, je výška trie reprezentujícího S rovna l (když S je neprázdná), a tedy vyhledání listu vyžaduje čas $O(l)$. Změna statusu vrcholu vyžaduje čas $O(k)$, protože $|\Sigma| = k$. Každý vrchol vyžaduje paměť velikosti k,listů je alespoň $|S|$ a každá cesta v kořene do listu reprezentujícího prvek z S má délku l, tedy vnitřních vrcholů může být až $O(l|S|)$. Shrňme tato pozorování.

Věta 9.1. *Algoritmus realizující operaci* **MEMBER** *v trie vyžaduje čas $O(l)$, algoritmy realizující operace* **INSERT** *a* **DELETE** *v trie vyžadují čas $O(kl)$. Datová struktura trie vyžaduje $\Omega(kl|S|)$ paměti.*

Nechť $\Sigma = \{0, 1, 2\}$, $l = 3$ a $S = \{102, 120, 121, 210, 211, 212\}$. Pak na obr. 4 je zobrazen trie reprezentující množinu S.



Obr. 4. Trie reprezentující S

Poznámka. Předpoklad, že univerzum je tvořeno slovy stejné délky, není omezující. Není-li splněn, lze to řešit dvěma způsoby. Můžeme přidat do abecedy písmeno 'mezera' a jím doplnit všechna slova na maximální délku. Druhá alternativa je, že rozšíříme proměnnou rep na všechny vrcholy stromu a test při vyhledávání a při doplňování trie v **INSERT**u pak nebude řízen podmínkou, že t není list, ale testem, zda i je menší nebo rovno délce slova α.

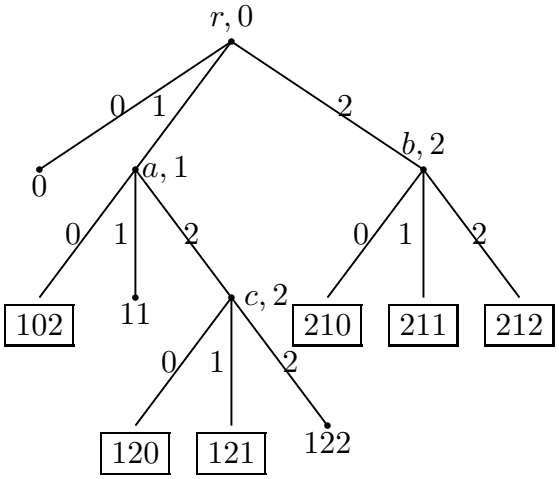
Poznámka. Všimněme si, že proměnná slovo(v) pro vrchol v sice byla použita při definici trie, ale není potřeba při jeho implementaci, proto ani není uvedena ve struktuře vrcholů trie.

Takto popsané trie mají malou časovou složitost, ale velký nárok na paměť. Je to způsobeno vrcholy v takovými, že slovo(v) je prefixem stejného počtu slov v S jako slovo(otec(v)). V takovém případě vrchol v přináší jen málo nové informace (pouze to, že jsme vyšetřili

další písmeno slova α). Analýza tohoto faktu vedla ke kompresi trie, kterou nyní popíšeme. Nejprve rozšíříme deklarace vrcholů stromu. Pro každý vrchol v označíme $\kappa(v)$ jeho hloubku v původním trie a pro každý list v budeme předpokládat, že je deklarována proměnná slovo(v). Pak, pokud je to možné, budeme na vrcholy trie aplikovat následující proces

(k) když existuje přesně jeden syn u vrcholu v takový, že buď u není list nebo rep(u) = $true$ (tedy všechny ostní synové u'jsou listy s rep(u') = $false$), pak odstraníme vrchol v a všechny jeho syny různé od u a vrcholem u nahradíme vrchol v.

Na obr. 5 je vidět komprese trie z obr. 4 (číslo u jména vnitřního vrcholu v je $\kappa(v)$ – jeho hloubka v původním trie, symbol u hrany odpovídá tomu, do kterého syna hrana vede, a u listů jsou příslušné hodnoty funkce slovo).



Obr. 5. Komprimovaný trie z Obr. 4

Algoritmy pro operace **MEMBER**, **INSERT** a **DELETE** pro komprimované trie je nutno modifikovat následujícím způsobem:

MEMBER-KT(α)
t :=kořen stromu
while t není list **do**
  $a := (\kappa(t)+1)$-ní písmeno slova α, $t :=$ a-tý syn t
enddo
if rep(t) = $true$ a slovo(t) = α **then** $\alpha \in S$ **else** $\alpha \notin S$ **endif**

INSERT-KT(α)
$t :=$ kořen stromu
while t není list **do**
  $a := (\kappa(t)+1)$-ní písmeno slova α, $t :=$ a-tý syn t
enddo
if slovo(t) není prefix α **then**
  $\beta :=$ nejdelší společný prefix α a slovo(t)
  najdi $a, b \in \Sigma$ takové, že βa je prefix α, βb je prefix slovo(t)

$v := t$
while $\kappa(v) \geq$ délka slova β **do** $v :=$ otec(v) **enddo**
$c := (\kappa(v)+1)$-ní písmeno β, $u := c$-tý syn v
Komentář: Vrchol u je takový, že β je vlastní prefix slova slovo(u).
vytvoř vrchol w, $\kappa(w) :=$ délka β
for every $x \in \Sigma \setminus \{b\}$ **do**
vytvoř nový list $z = x$-tý syn w
$\kappa(z) :=$ délka $\beta + 1$, slovo$(z) := \beta x$, rep$(z) := false$
enddo
c-tý syn $v := w$, b-tý syn $w := u$, $t := a$-tý syn w
endif
slovo$(t) := \alpha$, rep$(t) := true$

DELETE-KT(α)
$t :=$kořen stromu
while t není list **do**
$a := (\kappa(t)+1)$-ní písmeno slova α, $t := a$-tý syn t
enddo
if slovo$(t) = \alpha$ **then**
rep$(t) := false$, $u :=$ otec(t), $\kappa(t) := \kappa(u) + 1$, slovo$(t) :=$prefix slovo$(t)$ o délce $\kappa(t)$
if existuje právě jeden syn w vrcholu u, že w není list nebo rep$(w) =$ true **then**
odstraň všechny syny vrcholu u různé od w
$u := w$
endif
endif

Všimněme si, že vyhledávání slova α skončí v listu t, právě když se shodují $(\kappa(u)+1)$-ní písmena ve slovech α a slovo(t) pro všechny vrcholy u na cestě z kořene do listu t. Proto když $\alpha \in S$, algoritmus nalezne list t takový, že $\alpha =$ slovo(t), ale když $\alpha \notin S$, pak skončí v listu t takovém, že buď slovo$(t) \neq \alpha$, nebo slovo$(t) = \alpha$ a rep$(t) = false$. Tím je zajištěna korektnost těchto algoritmů. Navíc pro každý list t platí, že délka slovo(t) je $\kappa(t)$. Komprimovaný trie má nejvýše $|S| - 1$ vnitřních vrcholů, protože každý vnitřní vrchol má alespoň dva syny, které nejsou listy nebo jsou listy reprezentující prvky z S.

Nyní vypočteme očekávanou výšku komprimovaného trie reprezentujícího n-prvkovou množinu za předpokladu, že všechny n-prvkové množiny jsou stejně pravděpodobné. Označme q_d pravděpodobnost, že komprimovaný trie má výšku alespoň d. Pak očekávaná výška je

$$\sum_{d=1}^{\infty} d(q_d - q_{d+1}) = \sum_{d=1}^{\infty} q_d.$$

Proto budeme odhadovat velikost q_d. Všimněme si, že když

$$n = |S| = |\{\alpha \mid \alpha \text{ je prefix o délce } d \text{ nějakého slova v } S\}|,$$

pak výška komprimovaného trie reprezentujícího S je menší nebo rovna d. Protože n-prvkových množin slov délky d nad abecedou Σ je $\binom{k^d}{n}$ a protože n-tic slov o délce $l -$

d (nemusí být různá) je $(k^{l-d})^n = k^{n(l-d)}$, existuje $\binom{k^d}{n}k^{n(l-d)}$ množin S slov o délce l takových, že

$$n = |S| = |\{\alpha \mid \alpha \text{ je prefix o délce } d \text{ nějakého slova v } S\}|.$$

Odtud dostáváme, že

$$\begin{aligned}
q_d =& 1 - P(\text{výška je menší než } d) \leq \\
& 1 - \frac{\binom{k^{d-1}}{n}k^{n(l-d+1)}}{\binom{k^l}{n}} \leq 1 - \frac{\big(\prod_{i=0}^{n-1}(k^{d-1}-i)\big)k^{n(l-d+1)}}{k^{nl}} = 1 - \prod_{i=0}^{n-1}\frac{k^{d-1}-i}{k^{d-1}} = \\
& 1 - e^{\sum_{i=0}^{n-1}\ln(1-\frac{i}{k^{d-1}})} \leq 1 - e^{\sum_{i=0}^{n-1}\int_i^{i+1}\ln(1-\frac{x}{k^{d-1}})dx} = 1 - e^{\int_0^n \ln(1-\frac{x}{k^{d-1}})dx} = \\
& 1 - e^{(n-k^{d-1})\ln(1-\frac{n}{k^{d-1}})-n} \leq 1 - e^{-\frac{n^2}{k^{d-1}}} \leq \frac{n^2}{k^{d-1}}.
\end{aligned}$$

Vysvětlení: Zde první nerovnost plyne z monotonie pravděpodobnosti, druhá je (po rozepsání kombinačních čísel) zřejmá, třetí je důsledkem toho, že funkce $\ln(1-\frac{x}{k^{d-1}})$ je klesající, a proto $\ln(1-\frac{i}{k^{d-1}}) \geq \int_i^{i+1}\ln(1-\frac{x}{k^{d-1}})dx$. K výpočtu integrálu $\int_0^n \ln(1-\frac{x}{k^{d-1}})dx$ jsme použili substituci $1-\frac{x}{k^{d-1}} = t$ a snadno ověřitelný vzorec $\int \ln t dt = t\ln t - t$. Předposlední nerovnost jsme obdrželi ze vztahu $(n-k^{d-1})(\ln 1 - \frac{n}{k^{d-1}}) - n \geq -\frac{n^2}{k^{d-1}}$, který je důsledkem toho, že $n < k^{d-1}$ a že $x \geq \ln(1+x)$ v intervalu $(-1,\infty)$, a poslední nerovnost plyne z toho, že $e^x - 1 \geq x$ (pak $1 - e^x \leq -x$).

Položme $c = 2\lceil \log_k n \rceil$, pak $\frac{n^2}{k^c} \leq 1$. Tedy očekávaná výška stromu je

$$\sum_{d=1}^{\infty} q_d = \sum_{d=1}^{c} q_d + \sum_{d=c+1}^{\infty} q_d \leq \sum_{d=1}^{c} 1 + \sum_{d=c+1}^{\infty}\frac{n^2}{k^{d-1}} = c + \frac{n^2}{k^c}\Big(\sum_{d=0}^{\infty}\frac{1}{k^d}\Big) \leq 2\lceil \log_k n \rceil + \frac{1}{(1-\frac{1}{k})}.$$

Shrneme uvedené výsledky:

Věta 9.2. *Algoritmus realizující operaci* **MEMBER** *v komprimovaném trie vyžaduje v nejhorším případě čas $O(l)$ a v očekávaném případě (při rovnoměrném rozložení vstupních dat) čas $O(\log_k |S|)$. Algoritmy realizující operace* **INSERT** *a* **DELETE** *vyžadují v nejhorším případě čas $O(l+k)$ a v očekávaném případě čas $O(\log_k |S| + k)$. Komprimovaný trie má očekávanou výšku $O(\log_k |S|)$ a vyžaduje $O(k|S| + l|S|)$ prostoru.* □

Budeme-li uvažovat pouze operaci **MEMBER**, můžeme ještě dále a poměrně drasticky snížit nároky na paměť. Označme $V(T)$ množinu vnitřních vrcholů komprimovaného trie a ztotožněme list v se slovem slovo(v), když rep(v) = *true*, a se speciálním znakem NIL, když rep(v) = *false* (všimněme si, že pro list v takový, že rep(v) = *false*, potřebujeme znát slovo(v) jen v operacích **INSERT** a **DELETE**). Pak trie lze reprezentovat pomocí funkce κ (hloubka v původním trie) z $V(T)$ do přirozených čísel a matice M typu $V(T) \times \Sigma$, kde $M(v,a)$ je a-tý syn vrcholu v. Trie z Obr. 5 můžeme přepsat takto (algoritmus pro vyhledávání v této reprezentaci je zřejmý):

vrchol v	$\kappa(v)$
r	0
a	1
b	2
c	2

M	0	1	2
r	NIL	a	b
a	102	NIL	c
b	210	211	212
c	120	121	NIL

Při podrobnější analýze uvidíme, že místa, kde je NIL, nepřinášejí žádnou novou informaci (při vyhledávání mohou být nahrazena testem, zda přechod na další vrchol zvětší hodnotu funkce κ – pokud ne, pak $\alpha \notin S$). To znamená, že problém úsporné reprezentace trie můžeme redukovat na následující problém:

Je dána parciální matice M typu $r \times s$ (to znamená, že pro některá místa není hodnota prvku $M(i,j)$ definována a není ani podstatné, že není definována). Chceme matici M reprezentovat s co nejmenšími nároky na paměť tak, aby platilo, že když prvek $M(i,j)$ je definován, pak můžeme určit jeho hodnotu.

Pro zjednodušení popisu řešení budeme používat konvenci (pokud nebude řečeno jinak), že když vektor v má dimenzi k, pak jeho složky jsou $v(0)$, $v(1)$, až $v(k-1)$. Proto matice M v zadání úlohy bude mít řádky číslované $0, 1, \ldots, r-1$ a sloupce číslované $0, 1, \ldots, s-1$. Použijeme následující reprezentaci: matici M reprezentujeme vektorem ZAC dimenze r a vektorem val tak, že

> když $M(i,j)$ je definováno, pak $M(i,j) = val(ZAC(i)+j)$;
> když $M(i,j)$ a $M(i',j')$ jsou definovány a $(i,j) \neq (i',j')$, pak $ZAC(i)+j \neq ZAC(i')+j'$.

V tom případě můžeme matici M popisující komprimovaný trie z Obr. 5 reprezentovat těmito vektory (pro lepší pochopení souvislosti indexujeme složky vektoru ZAC vnitřními vrcholy trie, které píšeme do jeho horního řádku):

$$ZAC = \begin{pmatrix} r & a & b & c \\ 4 & 7 & 0 & 3 \end{pmatrix} \qquad val = (\,210 \quad 211 \quad 212 \quad 120 \quad 121 \quad a \quad b \quad 102 \quad NIL \quad c\,).$$

Je vidět, že tato reprezentace splňuje požadované podmínky a že dvě hodnoty NIL byly odstraněny. Problém je nalezení vektoru val s co nejmenší velikostí. Neformálně popíšeme algoritmus pro nalezení takové reprezentace.

KOMP-ŘAD(M)
1) V každém řádku i matice M spočteme počet míst (i,j), kde $M(i,j)$ je definováno, a setřídíme řádky podle této hodnoty.
2) Procházíme řádky v klesajícím pořadí podle počtu definovaných míst. Pro každý řádek i nalezneme nejmenší číslo $ZAC(i)$ (v rozsahu $0, 1, \ldots$) takové, že pro každý řádek i' předcházející řádku i a pro každé j a j' takové, že $M(i,j)$ a $M(i',j')$ jsou definovány, platí, že $ZAC(i)+j \neq ZAC(i')+j'$.

Je vidět, že vektor ZAC (a tím i vektor val) nemusí být pro daný komprimovaný trie určen jednoznačně. Záleží na tom, v jakém pořadí procházíme řádky se stejným počtem definovaných míst. V našem příkladě to bylo pořadí b, c, r, a.

Označme m počet dvojic (i,j) takových, že $M(i,j)$ je definováno, a pro každé l označme m_l počet dvojic (i,j) takových, že $M(i,j)$ je definováno a v řádku i existuje alespoň $l+1$ definovaných míst.

Věta 9.3. *Když pro každé l platí $m_l < \frac{m}{l+1}$, pak $ZAC(i) < m$ pro každý řádek i a algoritmus vyžaduje čas $O(rs+m^2)$.*

Důkaz. Pro každý řádek i určíme číslo r_i jako počet míst j takových, že $M(i,j)$ je definováno, a nalezneme seznam těchto míst. To vyžaduje čas $O(rs)$. K setřídění řádků podle počtu definovaných míst použijeme algoritmus **BUCKETSORT**. Ten vyžaduje čas $O(r+m)$, tedy krok 1) algoritmu vyžaduje čas $O(rs)$, protože $m \leq rs$. Pro každý řádek i potřebujeme ke zjištění, že $ZAC(i) = k$ je zakázaná hodnota, čas $O(r_i)$. Tedy když bude $ZAC(i) < m$ pro každý řádek, tak krok 2) bude vyžadovat čas $O(\sum_{i=0}^{r-1} mr_i) = O(m^2)$, protože $\sum_{i=0}^{r-1} r_i = m$, a čas celého algoritmu bude $O(rs+m^2)$.

Zbývá dokázat, že $ZAC(i) < m$ pro každý řádek i. Když aplikujeme krok 2) na řádek i takový, že $r_i = l$, pak vektor val obsahuje méně než m_{l-1} definovaných míst ($val(k)$ je definováno, právě když existuje dvojice (i',j') taková, že $M(i',j')$ je definováno, řádek i' předchází řádku i a $k = ZAC(i') + j'$, pak $val(k) = M(i',j')$). Aby hodnota $ZAC(i) = a$ byla zakázána, musí existovat (i,j) a (i',j') takové, že $M(i,j)$ a $M(i',j')$ jsou definovány, řádek i' předchází řádek i a $a + j = ZAC(i') + j'$. V tom případě řekneme, že dvojice (i,j) a (i',j') zakázaly hodnotu $ZAC(i) = a$. Tyto dvě dvojice však už žádnou jinou hodnotu $ZAC(i)$ nezakážou. Protože těchto dvojic je méně než lm_{l-1}, dostáváme z podmínky na matici M, že $lm_{l-1} \leq m$, a proto $ZAC(i) < m$. □

Podmínka na matice ve Větě 9.3 je příliš přísná, existuje jen málo matic, které ji splňují. Zdá se proto, že Věta 9.3 nemá praktický význam. Následující algoritmus nás přesvědčí o opaku. Jeho základní ideou je posunout sloupce v matici M tak, aby byly splněny předpoklady Věty 9.3. Uvažujme matici M, která vznikla z komprimovaného trie z Obr. 5, a posuňme poslední sloupec (odpovídající písmenu 2) o dva řádky. Tak vytvoříme matici M'

$$\begin{pmatrix} NIL & a & NIL \\ 102 & NIL & NIL \\ 210 & 211 & b \\ 120 & 121 & c \\ NIL & NIL & 212 \\ NIL & NIL & NIL \end{pmatrix}$$

a tu už můžeme kódovat předchozím algoritmem. Když posunutí sloupců bude zaznamenáno ve vektoru cd, pak matici M kódujeme do následujících tří vektorů:

$$cd = (\,0 \quad 0 \quad 2\,) \quad ZAC = (\,6 \quad 6 \quad 0 \quad 3 \quad 6 \quad 0\,)$$
$$val = (\,210 \quad 211 \quad b \quad 120 \quad 121 \quad c \quad 102 \quad a \quad 212\,).$$

V tomto případě platí, že když $M(i,j)$ je definováno, pak $M(i,j) = val(ZAC(cd(j)+i)+j)$, a když $M(i,j)$ a $M(i',j')$ jsou definovány pro různé (i,j) a (i',j'), pak $ZAC(cd(j)+i)+j \neq ZAC(cd(j')+i')+j'$. Vidíme, že jsme se takto zbavili NIL ve vektoru val, ale prodloužil se vektor ZAC. Je zřejmé, že kdybychom posunuli j-tý sloupec o $r(j-1)$ řádků, budou splněny předpoklady Věty 9.3, ale vektor ZAC bude mít dimenzi rs a to

znamená, že bychom neušetřili prostor. Takže naším cílem bude nalézt co nejmenší posunutí sloupců, aby byly splněny předpoklady algoritmu **KOMP-ŘAD**. Tento krok bude prvním krokem výsledného algoritmu, druhým krokem bude **KOMP-ŘAD**. K popisu prvního kroku budeme používat funkci $f(-,-)$ dvou proměnných, kterou budeme specifikovat později, a zavedeme následující označení: Pro $j = 0, 1, \ldots, s-1$ bude B_j označovat matici, která se skládá ze sloupců $0, 1, \ldots, j)$, které však pro $k = 0, 1, \ldots, j$ začínají až na $cd(k)$-tém řádku této matice (ostatní místa v matici B_j nejsou definována). Dále nechť $m(j)$ je počet definovaných míst v matici B_j a $m(j)_l$ počet definovaných míst v těch řádcích matice B_j, které mají alespoň $l+1$ definovaných míst. Pak první krok provede

Krok1
Postupně pro $j = 0, 1, \ldots, s-1$ nalezneme nejmenší číslo $cd(j)$ takové, že

$$m(j)_l \leq \frac{m}{f(l, m(j))} \qquad \text{pro každé } l = 0, 1, \ldots.$$

Protože $m(j) = m(j)_0$, musíme předpokládat, že $f(0, m(j)) \leq \frac{m}{m(j)}$ pro každé $j = 1, 2, \ldots, s$, jinak není v Kroku 1 splněna podmínka pro $l = 0$. Dále když pro každé l platí $f(l, m) \geq l + 1$, pak matice B_{s-1} splňuje předpoklady Věty 9.3, a to chceme. To znamená, že budeme požadovat, aby funkce $f(-,-)$ splňovala tyto okrajové podmínky:

$$f(0, m(j)) \leq \frac{m}{m(j)} \quad \text{pro každé } j = 1, 2, \ldots, s \quad \text{a} \quad f(l, m) \geq l + 1 \quad \text{pro každé } l = 0, 1, \ldots.$$

Nyní budeme analyzovat situaci, kdy při zpracování j-tého sloupce, je zakázána hodnota $cd(j) = a$. Víme, že v předcházejícím kroku byl splněn požadavek

$$m(j-1)_l \leq \frac{m}{f(l, m(j-1))} \qquad \text{pro každé } l = 0, 1, \ldots.$$

Dále $m(j)$ je počet definovaných míst v sloupcích $0, 1, \ldots, j$ matice M, a to je počet definovaných míst matice B_j při každé volbě (i zakázané) $cd(j)$. Předpokládejme, že $cd(j) = a$, a spočítejme $m(j)_l$ pro tuto matici. Protože hodnota $cd(j) = a$ je zakázaná, musí existovat l tak, že $m(j)_l > \frac{m}{f(l, m(j))}$. Když od této nerovnosti odečteme nerovnost pro l a $j-1$, dostaneme

$$m(j)_l - m(j-1)_l > \frac{m}{f(l, m(j))} - \frac{m}{f(l, m(j-1))}.$$

Tato nerovnost nám říka, kolik muselo přibýt definovaných míst v řádcích s alespoň $l+1$ definovaný mi místy v matici B_j proti matici B_{j-1}, aby podmínka pro l zakázala hodnotu $cd(j) = a$. Protože přidání sloupce j jen zvětší počet definovaných míst, platí $m(j)_l \geq m(j-1)_{l-1}$, a proto je vhodné předpokládat, že funkce f je klesající v druhé proměnné. Všimněme si, že $m(j)_l$ se může zvětšit jen tak, že buď k řádku matice B_{j-1} s alespoň $l+1$ definovanými místy přidá sloupec j další definované místo – pak se $m(j)_l$ zvětší o 1, nebo k řádku matice B_{j-1} s přesně l definovanými místy přidá sloupec j definované místo – pak se $m(j)_l$ zvětší o $l+1$. Tedy aby hodnota $cd(j) = a$ byla zakázána, musí $cd(j) = a$ vést k alespoň

$$\frac{\frac{m}{f(l, m(j))} - \frac{m}{f(l, m(j-1))}}{l+1}$$

dvojicím, kterými jsou řádek matice B_{j-1} s alespoň l definovanými místy a definované místo v sloupci j. Z předpokladů na matici B_{j-1} plyne, že B_{j-1} má nejvýše

$$\frac{m(j-1)_{l-1}}{l} \leq \frac{m}{lf(l-1,m(j-1))}$$

řádků s alespoň l definovanými místy, a sloupec j matice M má $m(j)-m(j-1)$ definovaných míst. Protože pro různé hodnoty $cd(j)$ se definovaná místa ve sloupci j matice M setkají s různými řádky matice B_{j-1} (jinak řečeno pro různé zakázan´ hodnoty $cd(j)$, jsou dvojice, které zavinily zvětšení $m(j)_l$ disjunktní), dostáváme, že splnění podmínky pro l zakáže nejvýše

$$\frac{\frac{(m(j)-m(j-1))m}{lf(l-1,m(j-1))}}{\frac{\frac{m}{f(l,m(j))}-\frac{m}{f(l,m(j-1))}}{l+1}} = \frac{(l+1)(m(j)-m(j-1))m}{lf(l-1,m(j-1))(\frac{m}{f(l,m(j))}-\frac{m}{f(l,m(j-1))})} =$$
$$\frac{l+1}{l}\frac{f(l,m(j-1))}{f(l-1,m(j-1))}\frac{m(j)-m(j-1)}{\frac{f(l,m(j-1))}{f(l,m(j))}-1}$$

hodnot $cd(j)$. Položme $l_0 = \min\{l \mid \frac{m}{f(l,m(j-1))} < l\}$. Pak pro každé $l \geq l_0$ platí, že $m(j-1)_l \leq \frac{m}{f(l,m(j-1))} < l$, a tedy neexistuje řádek v matici B_{j-1} s více než $l+1$ definovanými místy. Proto l_0 je poslední podmínka, která může zakázat nějakou hodnotu $cd(j)$, a odtud plyne, že zakázaných hodnot $cd(j)$ je nejvýše

$$\sum_{l=1}^{l_0}\frac{l+1}{l}\frac{m(j)-m(j-1)}{\frac{f(l,m(j-1))}{f(l,m(j))}-1}\frac{f(l,m(j-1))}{f(l-1,m(j-1))}.$$

Předpokládejme, že $f(x,y) = 2^{x(2-\frac{y}{m})}$. Pak $f(0,m(j)) = 2^0 = 1 \leq \frac{m}{m(j)}$ pro každé $j = 0,1,\ldots,s-1$. Dále $f(l,m) = 2^l \geq l+1$ pro každé $l = 0,1,\ldots,$ f je klesající v druhé proměnné, a tedy okrajové podmínky jsou splněny. Dosadíme hodnotu f do výše uvedeného součtu. Platí

$$\sum_{l=1}^{l_0}\frac{l+1}{l}\frac{m(j)-m(j-1)}{2^{l((2-\frac{m(j-1)}{m})-(2-\frac{m(j)}{m}))}-1}2^{(2-\frac{m(j-1)}{m})} \leq \sum_{l=1}^{l_0}\frac{l+1}{l}\frac{m(j)-m(j-1)}{2^{l\frac{m(j)-m(j-1)}{m}}-1}2^2 \leq$$
$$\sum_{l=1}^{l_0}4\frac{l+1}{l}\frac{m(j)-m(j-1)}{l\frac{m(j)-m(j-1)}{m}\ln 2} = \frac{4m}{\ln 2}\sum_{l=1}^{l_0}\frac{l+1}{l^2} \leq$$
$$\frac{4m}{\ln 2}(\sum_{l=1}^{l_0}\frac{1}{l}+\sum_{l=1}^{\infty}\frac{1}{l^2}) \leq \frac{4m}{\ln 2}(\ln l_0 + 1 + \frac{\pi^2}{6}) \leq 4m\log_2 l_0 + 15.3m,$$

protože $2^x - 1 = e^{x\ln 2} - 1 \geq x\ln 2$. Z $\frac{m}{2^{l(2-\frac{m(j-1)}{m})}} < l$ plyne, že $l_0 \leq \log_2 m$, a proto zakázaných hodnot $cd(j)$ je nejvýše $4m\log_2\log_2 m + 15.3m$. Tedy můžeme shrnout:

Věta 9.4. *Uvedený algoritmus reprezentuje parciální matici M typu $r \times s$ s m definovanými místy pomocí vektorů cd, ZAC a val tak, že*

(1) *když $M(i,j)$ je definováno, pak $M(i,j) = val(ZAC(cd(j)+i)+j)$,*
(2) *když $M(i,j)$ a $M(i',j')$ jsou definovány pro různé hodnoty (i,j) a (i',j'), pak*

$$ZAC(cd(j)+i)+j \neq ZAC(cd(j')+i')+j',$$

vektor cd má dimenzi s a hodnoty nejvýše rovné $4m\log\log m + 15.3m$, vektor ZAC má dimenzi menší než $4m\log\log m + 15.3m + r$ a hodnoty menší než m a vektor val má dimenzi menší než $m+s$. Algoritmus vyžaduje čas $O(s(r+m\log\log m)^2)$.

Důkaz. Zbývá odhadnout čas procedury **Krok1**. Když jsou uchovány hodnoty $m(j-1)$ a $m(j-1)_l$, pak pro dané $cd(j) = a$ je na nalezení nových hodnot $m(j)_l$ třeba čas $O(r + m\log\log m)$. Tedy na ověření, zda je možná hodnota $cd(j) = a$, je zapotřebí $O((r+m\log\log m)+l_0) = O((r+m\log\log m)+\log m)$ času. Protože zakázaných hodnot je nejvýše $4m\log\log m + 15.3m$, dostáváme požadovaný odhad na dobu výpočtu algoritmu. □

Protože M má jen m definovaných míst, dostáváme z algoritmu **KOMP-ŘAD**, že nejvýše m hodnot vektoru ZAC je různých od nuly. Tohoto faktu využijeme pro další kompresi vektoru ZAC. Budeme řešit následující problém:

Předpokládejme, že máme reprezentovat vektor v o dimenzi nd, kde n a d jsou přirozená čísla, takový, že $i_0 < i_1 < \cdots < i_{s-1}$ je rostoucí posloupnost všech indexů j, kde $v(j) \neq 0$. Vytvořme tedy vektor cv o dimenzi s takový, že $cv(j) = v(i_j)$ pro každé $j = 0,1,\ldots,s-1$. Problém je, jak pro danné i spočítat $v(i)$. K tomu použijeme další dva vektory. Nejprve vytvoříme vektor $base$ o dimenzi n tak, že $base(i) = -1$, když $i_j \div d \neq i$ pro každé $j = 0,1,\ldots,s-1$, a jinak $base(i)$ je nejmenší l takové, že $i_l \div d = i$. (Připomínáme, že $\div$ je celočíselné dělení, což znamená, že je definováno jen pro celá čísla a, b, výsledek c je také celé číslo a $a \div b = c$, právě když existuje přirozené číslo q takové, že $q < |b|$ a $a = bc + q$, tedy $a \equiv q \bmod |b|$ a c je určeno jednoznačně.) Dále vytvořme vektor $offset(i,j)$, kde $i = 0,1,\ldots,n-1$ a $j = 0,1,\ldots,d-1$, takový, že $offset(i,j) = -1$, když $v(id+j) = 0$, a $offset(i,j) = l - base(i)$, když $id + j = i_l$. Při výpočtu $v(i)$ pak postupujeme takto: spočítáme $k = i \div d$, $l = i \bmod d$ a položíme $v(i) = 0$, když $offset(k,l) = -1$, a jinak $v(i) = cv(offset(k,l) + base(k))$. Pak hodnoty $offset(k,l)$ jsou v rozmezí od -1 do $d-1$ pro každé $k = 0,1,\ldots,n-1$ a $l = 0,1,\ldots,d-1$. Toho využijeme a definujeme vektor off dimenze n tak, že

$$off(k) = \sum_{l=0}^{d-1} (offset(k,l)+1)(d+1)^l.$$

Pak $offset(k,l) = ((off(k) \div (d+1)^l) \bmod (d+1)) - 1$ a $off(k) \leq (d+1)^d$ pro každé $k = 0,1,\ldots,n-1$. Tedy dostáváme

Věta 9.5. *Když reprezentujeme vektor v třemi vektory cv, $base$ a off, pak $base$ a off mají dimenzi n, hodnoty $base$ jsou od -1 do s a hodnoty off jsou od 0 do $(d+1)^d - 1$. Hodnota $v(i)$ se spočítá v čase $O(1)$.*

Poznámka. Je-li většina hodnot vektoru v rovna nějaké konstantě c, pak uložíme tuto hodnotu c a nalezneme rostoucí posloupnost všech indexů j takových, že $v(j) \neq c$. Zbytek už je stejný.

Všimněme si, že když $d \leq \lceil \log\log n \rceil$, pak $(d+1)^d < n$, a tedy dostáváme:

Důsledek 9.6. *Trie reprezentující množinu S lze uložit pomocí 5 vektorů o dimenzi $|S|$ s hodnotami menšími než $4|S|\log\log(|S|) + 15.3|S|$.*

Reprezentace trie z Důsledku 9.6 nepodporuje operace **INSERT** a **DELETE**. Vektory by se po každém úspěšném provedení operace musely znovu přepočítat. Nalézt optimální uložení trie je Δ_2-úplný problém.

Trie zavedl a poprvé vyšetřoval Fredkin 1960. Komprimované trie a jejich vztah k řídkým maticím poprvé formuloval Ziegler. Výsledky o reprezentaci parciálních matic vektorů pocházejí od Tarjana a Yao 1979.

10. Dynamizace

V průběhu našeho dosavadního pojednání o datových strukturách jsme si mohli všimnout, že ne všechny umožňují stejně efektivní provedení všech základních operací. Např. datová struktura uspořádané pole má sice řadu algoritmů pro operaci **MEMBER**, ale s efektivní realizací operací **INSERT** a **DELETE** jsou potíže a teprve až její zobecnění na binární vyhledávací stromy (např. AVL-stromy, červeno-černé stromy atd.) umožnilo nalézt efektivní algoritmy pro tyto operace. Nejsou však známé žádná přirozená a jednoduchá zobecnění pro perfektní hašování nebo pro reprezentaci komprimovaných trie pomocí matic. Lze se dohadovat, že je to způsobeno tím, že nalézt vhodnou reprezentaci v těchto strukturách je samo o sobě velmi složité. Ale tato situace je běžná v datových strukturách pro vícedimenzionální vyhledávání nebo ve výpočetní geometrii. To vedlo k hledání obecných metod pro konstrukci algoritmů realizujících operace **INSERT** a **DELETE**. Prvním krokem tímto směrem je obecná definice vyhledávacího problému.

Mějme tři univerza U_1, U_2 a U_3. Pak funkce $f : U_1 \times 2^{U_2} \to U_3$ se nazývá vyhledávací problém. Řekneme, že $\mathcal{S}$ je statická datová struktura řešící vyhledávací problém f, když je dán algoritmus, který pro množinu $A \subseteq U_2$ zkonstruuje datovou strukturu $\mathcal{S}(A)$, a algoritmus závislý na A, který pro $x \in U_1$ a $\mathcal{S}(A)$ vyčíslí $f(x, A)$. Struktura se nazývá semidynamická, když navíc existuje algoritmus, který pro $x \in U_2$ a pro $\mathcal{S}(A)$ zkonstruuje $\mathcal{S}$-strukturu $\mathcal{S}(A\cup\{x\})$, a algoritmus vyčíslující $f(y, A\cup\{x\})$ pro každé $y \in U_1$ (tato operace se nazývá **INSERT**(x)). Když ještě navíc existuje algoritmus, který pro $x \in U_2$ a $\mathcal{S}(A)$ zkonstruuje $\mathcal{S}(A \setminus \{x\})$, a algoritmus vyčíslující$f(y, A \setminus \{x\})$ pro každé $y \in U_1$ (tato operace je **DELETE**(x)), pak se struktura nazývá dynamická. Naším cílem je nalézt metody, které změní statickou strukturu řešící vyhledávací problém f na semidynamickou nebo dynamickou strukturu řešící f a pochopitelně chceme, aby tyto struktury byly efektivní.

Nejprve několik příkladů:

- (•) Položme $U_1 = U_2 = U$, $U_3 = \{0, 1\}$, $f(x, A) = 1$, když $x \in A$, a $f(x, A) = 0$, když $x \notin A$. Pak operace **MEMBER** je vyhledávací problém.
- (•) Nechť $U_1 = U_2 =$ Euklidovský prostor, $U_3 =$ množina nezáporných reálných čísel, $f(x, A)$ je vzdálenost x od A. Pak určení vzdálenosti v Euklidovských prostorech je vyhledávací problém.
- (•) Nechť $U_1 = U_2 =$ Euklidovská rovina, $U_3 = \{0, 1\}$, $f(x, A) = 0$, když x nepatří do konvexního obalu množiny A, $f(x, A) = 1$, když x je prvek konvexního obalu množiny A. Pak příslušnost ke konvexnímu obalu v rovině je vyhledávací problém.
- (•) Nechť $U_1 = U_2 = U_3 = U$, kde U je konečné totálně uspořádané univerzum s nejmenším prvkem, $f(x, A) = \max\{a \in A \mid a \le x\}$ (když $x < \min A$, pak $f(x, A)$ je nejmenší prvek v U). Pak nalezení předchůdce v množině je vyhledávací problém.

Náš návrh obecné metody na semidynamizaci je založen na paradigmatu rozděl a panuj, přesněji na ideách, které vedly k binomiálním haldám. Proto naši metodu nelze použít na každý vyhledávací problém. Řekneme, že vyhledávací problém je rozložitelný, když existuje binární operace $\odot$ na univerzu U_3 taková, že pro disjunktní množiny $A, B \subseteq U_2$ a pro $x \in U_1$ platí

$$f(x, A) \odot f(x, B) = f(x, A \cup B).$$

Zřejmě operace **MEMBER**, problém nalezení vzdálenosti v Euklidovských prostorech a problém nalezení předchůdce jsou rozložitelné problémy. V prvním případě $\odot = \vee$, v druhém $\odot = \min$ a ve třetím $\odot = \max$. Příslušnost ke konvexnímu obalu není rozložitelný problém.

V následujícím textu předpokládáme, že f je rozložitelný vyhledávací problém a $\mathcal{S}$ je statická struktura řešící f. Aby náš návrh byl efektivní, předpokládáme, že operace $\odot$ je vyčíslitelná v čase $O(1)$. Abychom mohli posoudit efektivnost návrhu, označíme $Q_{\mathcal{S}}$ čas na vyčíslení $f(x, A)$ (to znamená, že čas pro vyčíslení $f(x, A)$, kde $|A| \leq n$, je nejvýše $Q_{\mathcal{S}}(n)$), $S_{\mathcal{S}}$ paměťovou náročnost $\mathcal{S}$ (tj. paměť potřebná k uložení $\mathcal{S}(A)$, kde $|A| \leq n$, je nejvýše $S_{\mathcal{S}}(n)$) a $P_{\mathcal{S}}$ čas na konstrukci $\mathcal{S}(A)$ (tj. čas potřebný na konstrukci $\mathcal{S}(A)$ pro $|A| \leq n$ je nejvýše $P_{\mathcal{S}}(n)$). Budeme předpokládat, že funkce $Q_{\mathcal{S}}(n)$, $\frac{S_{\mathcal{S}}(n)}{n}$ a $\frac{P_{\mathcal{S}}(n)}{n}$ jsou neklesající. Funkce $\frac{S_{\mathcal{S}}(n)}{n}$ dává odhad na velikost paměti, která je potřebná na jeden prvek v $\mathcal{S}(A)$, a funkce $\frac{P_{\mathcal{S}}(n)}{n}$ dává vhodný dolní odhad času potřebného pro operaci **INSERT**. Proto je přirozené (nikoliv nutné) předpokládat neklesajícnost těchto funkcí.

Nejprve navrhneme semidynamickou strukturu řešící vyhledávací problém f a ukážeme, že amortizovaná složitost operace **INSERT** je blízká funkci $\frac{P_{\mathcal{S}}(n)}{n}$ (zde odhad na amortizovanou složitost říká, že pro dostatečně velké m je čas na provedení posloupnosti m operací **INSERT** menší než $m-$násobek odhadu amortizované složitosti).

Konstrukce 10.1. Mějme danou množinu $A \subseteq U_2$. Nechť je zvolen soubor disjunktních množin $\{A_i \mid i = 0, 1, \dots\}$ takový, že $A_i \subseteq A$, $A = \bigcup_{i=0}^{\infty} A_i$ a buď $A_i = \emptyset$ nebo $|A_i| = 2^i$ pro každé $i = 0, 1, \dots$. Z binárního zápisu čísla plyne, že taková posloupnost množin existuje a navíc $A_i = \emptyset$ pro každé $i > 1 + \log_2 |A|$. Struktura $\mathcal{P}(A)$ se bude skládat ze seznamu $\mathcal{S}$-struktur $\{\mathcal{S}(A_i) \mid i = 0, 1, \dots, 1 + \lfloor \log |A| \rfloor, A_i \neq \emptyset\}$, ze seznamu $\{s(A_i) \mid i = 0, 1, \dots, 1 + \lfloor \log |A| \rfloor, A_i \neq \emptyset\}$ prostých dvousměrných spojových seznamů $s(A_i)$ prvků množin A_i a ze struktury T reprezentující A (např. červeno-černý strom, (a, b)-strom apod.). Popíšeme algoritmy pro strukturu $\mathcal{P}$. Nejprve algoritmus pro vyčíslení $f(x, A)$.

VYCIS(x)
zvol $i_0 \in \{0, 1, \dots, 1 + \lfloor \log |A| \rfloor\}$ takové, že $A_{i_0} \neq \emptyset$
vypočti $f(x, A_{i_0})$, $f(x, A) := f(x, A_{i_0})$
for every $i = 0, 1, \dots, 1 + \lfloor \log |A| \rfloor$ takové, že $A_i \neq \emptyset$ a $i \neq i_0$ **do**
  vypočti $f(x, A_i)$, $f(x, A) := f(x, A) \odot f(x, A_i)$
enddo

Korektnost algoritmu pro vyčíslení $f(x, A)$ plyne okamžitě z předpokladů.

Následující algoritmus implementuje operaci **INSERT**(x).

INSERT(x)
if x není reprezentován v T **then**

INSERT(x, T) (Komentář: vkládáme x do pomocné struktury T)
najdi nejmenší i takové, že $A_i = \emptyset$
$s(A_i) := \{x\}$
for every $j < i$ **do**
$s(A_i) :=$ konkatenace $s(A_i)$ a $s(A_j)$, zruš $s(A_j)$ a $\mathcal{S}(A_j)$
enddo
vytvoř strukturu $\mathcal{S}(A_i)$ pro množinu $A_i = \{x\} \cup \bigcup_{j<i} A_j$ pomocí seznamu $s(A_i)$
(Komentář: po skončení výpočtu $A_i = \{x\} \cup \bigcup_{j<i} A_j$ se pro $j < i$ množiny A_j vyprázdní)
endif

Protože $2^i = 1 + \sum_{j=0}^{i-1} 2^j$, dostáváme, že algoritmus pro **INSERT** je korektní.

Poznámka. Všimněme si, že stejně jako u binomiálních hald $|A|$ jednoznačně určuje i takové, že $A_i \neq \emptyset$. Je to místo v binárním rozvoji čísla $|A|$, kde je 1.

Věta 10.2. *Struktura $\mathcal{P}$ je semidynamická struktura řešící f. Algoritmus na vyčíslení f vyžaduje čas $O(Q_{\mathcal{S}}(n)\log(n))$, struktura $\mathcal{P}$ potřebuje $O(S_{\mathcal{S}}(n))$ paměti a amortizovaná složitost algoritmu implementujícího operaci* **INSERT** *je $O(\frac{P_{\mathcal{S}}(n)\log(n)}{n})$.*

Když $Q_{\mathcal{S}}(n) = n^\varepsilon$ pro nějaké $\varepsilon > 0$, pak algoritmus na vyčíslení f ve struktuře $\mathcal{P}$ vyžaduje čas $O(n^\varepsilon)$.

Když $P_{\mathcal{S}}(n) = n^\varepsilon$ pro nějaké $\varepsilon > 1$, pak amortizovaná složitost algoritmu pro operaci **INSERT** *ve struktuře $\mathcal{P}$ je $O(n^{\varepsilon-1})$.*

Důkaz. Z Konstrukce 10.1 plyne, že $\mathcal{P}$ je korektně definovaná semidynamická struktura řešící f. Zbývá odhadnout její složitost.

Čas na výpočet $f(x, A)$, když $|A| = n$, je

$$\log_2 n + \sum\{Q_{\mathcal{S}}(|A_i|) \mid A_i \neq \emptyset\} \leq \log_2 n + \sum_{i=0}^{1+\lfloor \log n \rfloor} Q_{\mathcal{S}}(n) = (Q_{\mathcal{S}}(n) + 1)(1 + \lfloor \log n \rfloor) = O(Q_{\mathcal{S}}(n) \log n),$$

protože výpočet $f(x, A_i)$ vyžaduje $O(Q_{\mathcal{S}}(|A_i|) = O(Q_{\mathcal{S}}(n))$ času, vyčíslení $\odot$ vyžaduje čas $O(1)$ a je nejvýše $\log n$ čísel i takových, že $A_i \neq \emptyset$. Když $Q_{\mathcal{S}}(n) = |S|^\varepsilon$ pro $\varepsilon > 0$, pak

$$\log_2 n + \sum\{Q_{\mathcal{S}}(|A_i|) \mid A_i \neq \emptyset\} \leq \log_2 n + \sum_{i=0}^{1+\lfloor \log n \rfloor} (2^i)^\varepsilon = \log_2 n + \frac{2^{(2+\lfloor \log n \rfloor)\varepsilon} - 1}{2^\varepsilon - 1} = O(n^\varepsilon).$$

Odhad paměti:

$$cn + \sum\{S_{\mathcal{S}}(|A_i|) \mid A_i \neq \emptyset\} \leq cn + \sum_{i=0}^{1+\lfloor \log n \rfloor} \frac{S_{\mathcal{S}}(2^i)}{2^i} 2^i \leq cn + \sum_{i=0}^{1+\lfloor \log n \rfloor} \frac{S_{\mathcal{S}}(n)}{n} 2^i = cn + \frac{S_{\mathcal{S}}(n)}{n} \sum_{i=0}^{1+\lfloor \log n \rfloor} 2^i \leq cn + \frac{S_{\mathcal{S}}(n)}{n} 2n = cn + 2S_{\mathcal{S}}(n) = O(S_{\mathcal{S}}(n)),$$

protože z předpokladu, že $\frac{S_{\mathcal{S}}(n)}{n}$ je neklesající, plyne $\frac{S_{\mathcal{S}}(2^i)}{2^i} \leq \frac{S_{\mathcal{S}}(n)}{n}$ pro $i = 0, 1, \ldots, 1 + \lfloor \log |A| \rfloor$ a také, že $S_{\mathcal{S}}(n) \geq n$.

Amortizovaná složitost operace **INSERT**: Pro výpočet amortizované složitosti použijeme potenciálovou metodu. Test, zda $x \in A$, a případné přidání x do množiny reprezentované T vyžaduje čas $O(\log n)$. Protože konkatenace dvou seznamů vyžaduje čas $O(1)$ a protože konkatenujeme nejvýše $1 + \log n$ seznamů, tak vytvoření $s(A_i)$ a zrušení přebytečných struktur vyžaduje čas $O(\log n)$. Aby se vytvořila $\mathcal{S}$-struktura pro A_i, musí před operací **INSERT** platit, že $|A_j| = 2^j$ pro každé $j = 0, 1, \ldots, i-1$ a $A_i = \emptyset$, a po provedení operace **INSERT** platí, že $A_j = \emptyset$ pro všechna $j < i$ a $|A_i| = 2^i$.

Zavedeme potenciálovou funkci Φ. Pro množinu $A \subseteq U_2$ a pro přirozené číslo definujme funkci

$$\Phi_i(A) = \begin{cases} 0 & \text{když } A_i \neq \emptyset, \\ \sum_{j=0}^{i-1} \frac{P_{\mathcal{S}}(2^i)}{2^i} |A_j| & \text{když } A_i = \emptyset. \end{cases}$$

Potenciál je $\Phi(A) = \sum_{i=0}^{1+\lfloor \log |A| \rfloor} \Phi_i(A)$.

V operaci **INSERT**(x) nejprve pomocí struktury T testujeme, zda $x \in A$. Pokud $x \in A$, operace končí a strukturu nemění a tedy amortizovaná složitost je stejná jako časová složitost (tj. $O(\log |A|)$). Když $x \notin A$ nalezneme nejmenší i takové, že $A_i = \emptyset$, spojením všech seznamů pro A_j, $j < i$ a přidáním x vytvoříme i-tý seznam pro novou množinu, kterou budeme značit B. To vyžaduje čas $O(\log |A| + P_{\mathcal{S}}(2^i))$. Tedy $B_i = \bigcup_{j=0}^{i} A_i \cup \{x\}$, $B_j = \emptyset$ pro $j < i$ a $B_j = A_j$ pro $j > i$. Proto

$$\Phi_k(A) = \begin{cases} 0 & \text{když } k < i, \\ \frac{P_{\mathcal{S}}(2^i)}{2^i}(2^i - 1) & \text{když } k = i \end{cases}$$

$$\Phi_k(B) = \begin{cases} 0 & \text{když } k \leq i \text{ nebo } B_k \neq \emptyset, \\ \Phi_k(A) + \frac{P_{\mathcal{S}}(2^k)}{2^k} & \text{když } k > i \text{ a } B_k = \emptyset. \end{cases}$$

První vztah plyne z toho, že $A_k \neq \emptyset$ pro $k < i$, druhý vztah plyne z toho, že $\sum_{j=0}^{i-1} |A_j| = \sum_{j=0}^{i-1} 2^j = 2^i - 1$, třetí vztah plyne z toho, že $B_j = \emptyset$ pro $j < i$ a čtvrtý vztah plyne z toho, že pro $k > i$ platí $\sum_{j=0}^{k-1} |B_j| = 1 + \sum_{j=0}^{k-1} |A_j|$. Odtud plyne, že

$$\Phi(B) - \Phi(A) \leq \sum_{k=i+1}^{1+\lfloor \log |A| \rfloor} \frac{P_{\mathcal{S}}(2^k)}{2^k} - \frac{P_{\mathcal{S}}(2^i)}{2^i}(2^i - 1).$$

Tedy amortizovanou složitost můžeme odhadnout

$$\log(|A|) + P_{\mathcal{S}}(2^i) - \frac{P_{\mathcal{S}}(2^i)}{2^i}(2^i - 1) + \sum_{j>i, A_j=\emptyset} \frac{P_{\mathcal{S}}(2^j)}{2^j} \leq$$

$$\log(|A|) + \sum_{j=0}^{1+\lfloor \log |A| \rfloor} \frac{P_{\mathcal{S}}(2^j)}{2^j} \leq$$

$$\log(|A|) + (1 + \lfloor \log |A| \rfloor) \frac{P_{\mathcal{S}}(|A|)}{|A|} =$$

$$O(\log |A| \frac{P_{\mathcal{S}}(|A|)}{|A|}),$$

kde jsme použili, že $\frac{P_{\mathcal{S}}(n)}{n}$ je neklesající funkce (tedy $\frac{P_{\mathcal{S}}(2^i)}{2^i} \leq \frac{P_{\mathcal{S}}(|A|)}{|A|}$ pro $i < 1 + \lfloor \log |A| \rfloor$). Tím jsme dokázali požadovaný odhad složitosti struktury $\mathcal{P}$.

Když $P_{\mathcal{S}}(n) = n^{\varepsilon}$ pro $\varepsilon > 1$, pak

$$c \log n + \sum_{i=0}^{1+\lfloor \log n \rfloor} \frac{P_{\mathcal{S}}(2^i)}{2^i} \leq c \log n + \sum_{i=0}^{1+\lfloor \log n \rfloor} \frac{(2^i)^{\varepsilon}}{2^i} = O(n^{\varepsilon - 1}).$$

Tím jsme dokázali požadovaný odhad složitosti struktury $\mathcal{P}$. □

Neformálně řečeno, když vytváříme $\mathcal{S}$-strukturu pro množinu A_i, tak před operací byly $A_j \neq \emptyset$ pro $j < i$ a $A_i = \emptyset$ a po skončení operace jsou $A_j = \emptyset$ pro $j < i$ a $A_i \neq \emptyset$. Proto abychom příště vytvářeli $\mathcal{S}$-strukturu pro množinu A_i, musíme před touto operací provést $2^i - 1$ úspěšných operací **INSERT** (tj. operací které přidávaly prvek), aby se naplnily množiny A_j pro $j < i$, pak další úspěšnou operaci **INSERT**, po jejímž provedení budou množiny $A_j = \emptyset$ pro $j \leq i$, a pak znovu $2^i - 1$ úspěšných operací**INSERT**, aby $A_j \neq \emptyset$ pro $j < i$, a teprve následující úspěšná operace **INSERT** vytváří znovu $\mathcal{S}$-strukturu pro A_i. Tedy neformálně řečeno, $\mathcal{S}$-strukturu pro A_i vytváří jen jedna z 2^{i+1} úspěšných operací **INSERT**. Tento fakt formálně popisuje tento potenciál.

Protože amortizovaná složitost není vždy uspokojující (odhad amortizované složitosti je vhodný pro dávkové zpracování požadavků, zatímco pro interaktivní zpracování je důležitý odhad v nejhorším případě), hledaly se další metody, které by vedly k algoritmu pro operaci **INSERT**, který by měl dobrý odhad složitosti v nejhorším případě. Základní idea je rozložit budování $\mathcal{S}$-struktury pro množinu velikosti 2^i do 2^i po sobě jdoucích **INSERT**ů. Problém je, že $\mathcal{S}$-struktury musí pokrývat celou množinu A, jinak nejsme schopni vyřešit vyhledávací problém f. Řešení spočívá v tom, že budeme mít více množin velikosti 2^i, pro které je vybudovaná $\mathcal{S}$-struktura. Pak, když máme vybudované $\mathcal{S}$-struktury pro dvě množiny $A_{i,0}$ a $A_{i,1}$ velikosti 2^i, začneme budovat $\mathcal{S}$-strukturu pro množinu $A_{i,0} \cup A_{i,1}$ velikosti 2^{i+1} a teprve až tuto $\mathcal{S}$-strukturu vytvoříme, zrušíme $\mathcal{S}$-struktury pro množiny $A_{i,0}$ a $A_{i,1}$. Nyní popíšeme formálně návrh semidynamické struktury $\mathcal{P}_1$.

Konstrukce 10.3. Mějme množinu $A \subseteq U_2$, pro kterou chceme vytvořit $\mathcal{P}_1$-strukturu. Předpokládejme, že máme množiny $A_{i,j}$ pro $i = 0, 1, \ldots 1 + \lfloor \log |A| \rfloor$, $j = 0, 1, \ldots, k_i$, kde pro každé i je $k_i \in \{-1, 0, 1, 2, 3\}$ a platí

(a) když $k_i \geq 0$, pak $\emptyset \neq A_{i,j} \subseteq A$ a $|A_{i,j}| = 2^i$ pro každé $j = 0, 1, \ldots, k_i$ (tedy $k_i = -1$, právě když $A_{i,j} = \emptyset$ pro každé j, a to je právě když $A_{i,k_i} = \emptyset$);

(b)
$$\{A_{0,j} \mid j = 0, 1, \ldots, k_0\} \cup \{A_{i,j} \mid k_i > 0,\ j = 0, 1, \ldots, k_i - 1\}$$
je rozklad množiny A;

(c) když $k_i \geq 0$ pro $i > 0$, pak $k_{i-1} \geq 1$ a $A_{i,k_i} = A_{i-1,0} \cup A_{i-1,1}$;

(d) pro každé $j = 0, 1, \ldots, k_0$ je pro množinu $A_{0,j}$ vytvořena $\mathcal{S}$-struktura $\mathcal{S}(A_{0,j})$;

(e) pro každé $i > 0$ takové, že $k_i \geq 0$, a pro každé $j = 0, 1, \ldots, k_i - 1$ je pro množinu $A_{i,j}$ vytvořena $\mathcal{S}$-struktura $\mathcal{S}(A_{i,j})$ a pro množinu A_{i,k_i} je částečně vytvořena $\mathcal{S}$-struktura $\mathcal{S}(A_{i,k_i})$, ale ta není dokončena (říkáme, že $\mathcal{S}$-struktura $\mathcal{S}(A_{i,k_i})$ je rozpracovaná).

Struktura $\mathcal{P}_1$ je pak tvořena seznamem $\mathcal{S}$-struktur a rozpracovaných $\mathcal{S}$-struktur

$$\{\mathcal{S}(A_{i,j}) \mid k_i \geq 0,\ j = 0, 1, \ldots, k_i\},$$

dále seznamem prostých dvousměrných spojových seznamů prvků množin $A_{i,j}$

$$\{s(A_{0,j}) \mid j = 0, 1, \ldots, k_0\} \cup \{s(A_{i,j}) \mid k_i \geq 0,\ j = 0, 1, \ldots, k_i - 1\}$$

a datovou strukturou T reprezentující množinu A. Následující algoritmus vyčíslí $f(x, A)$.

VYCIS(x)
vypočti $f(x, A_{0,k_0})$, $f(x, A) := f(x, A_{0,k_0})$
for every i takové, že $k_i \geq 0$ **do**
  for every $j = 0, 1, \ldots, k_i - 1$ **do**
    vypočti $f(x, A_{i,j})$, $f(x, A) = f(x, A) \odot f(x, A_{i,j})$
  enddo
enddo

Korektnost algoritmu pro vyčíslení $f(x, A)$ plyne okamžitě z předpokladů (c), (d) a (e). Následující algoritmus implementuje operaci **INSERT**(x).

INSERT(x)
if x není reprezentován datovou strukturou T **then**
  INSERT(x, T) (Komentář: vkládáme x do pomocné struktury T)
  $A_{0,k_0+1} = \{x\}$, $k_0 := k_0 + 1$
  vytvoř $\mathcal{S}$-strukturu $\mathcal{S}(A_{0,k_0})$ a spojový seznam $s(A_{0,k_0})$ pro množinu A_{0,k_0}
  $i := 1$
  while $k_i \geq 0$ **do**
    proveď dalších $\frac{P_{\mathcal{S}}(2^i)}{2^i}$ kroků v konstrukci $\mathcal{S}$-struktury $\mathcal{S}(A_{i,k_i})$
    if konstrukce $\mathcal{S}$-struktury $\mathcal{S}(A_{i,k_i})$ je skončena **then**
      $s(A_{i,k_i})$:= konkatenace $s(A_{i-1,0})$ a $s(A_{i-1,1})$
      zruš $\mathcal{S}$-struktury $\mathcal{S}(A_{i-1,0})$, $\mathcal{S}(A_{i-1,1})$ a spojové seznamy $s(A_{i-1,0})$, $s(A_{i-1,1})$
      for every $j = 2, 3, \ldots, k_{i-1}$ **do**
        $\mathcal{S}(A_{i-1,j-2}) := \mathcal{S}(A_{i-1,j})$, $s(A_{i-1,j-2}) := s(A_{i-1,j})$
      enddo
      $k_{i-1} := k_{i-1} - 2$, $k_i := k_i + 1$
      proveď první krok konstrukce $\mathcal{S}(A_{i,k_i})$ pro $A_{i,k_i} = A_{i-1,0} \cup A_{i-1,1}$
(Komentář: prvky A_{i,k_i} se berou ze seznamů $s(A_{i-1,0})$ a $s(A_{i-1,1})$)
    endif
    $i := i + 1$
  enddo
  if $k_{i-1} = 1$ **then**
    $k_i = 0$
    proveď první krok konstrukce $\mathcal{S}(A_{i,0})$ pro $A_{i,0} = A_{i-1,0} \cup A_{i-1,1}$
(Komentář: prvky $A_{i,0}$ se berou ze seznamů $s(A_{i-1,0})$ a $s(A_{i-1,1})$)
  endif
endif

Korektnost algoritmu plyne z předpokladů. Všimněme si, že když skončila konstrukce $\mathcal{S}$-struktury $\mathcal{S}(A_{i,k_i})$, pak běh cyklu pro $i-1$ skončil také konstrukci $\mathcal{S}$-struktury $\mathcal{S}(A_{i-1,k_{i-1}})$.

V tomto okamžiku může platit $k_i = 4$, ale další běh cyklu (pro $i+1$) zmenší k_i na 2. Proto je algoritmus korektní.

Korektnost struktury $\mathcal{P}_1$ je ukázána v Konstrukci 10.3. Protože i je v rozmezí od 0 do $\log n$ a $j \in \{0,1,2,3\}$, dostáváme, že odhady na dobu výpočtu f a na paměťovou náročnost se od odhadů pro strukturu $\mathcal{P}$ liší jen o multiplikativní konstantu. Protože při úspěšné operaci **INSERT** se provede jen $O(\frac{P_{\mathcal{S}}(2^i)}{2^i})$ kroků pro vybudování $\mathcal{S}$-struktury $\mathcal{S}(A_{i,k_i})$, tak čas operace **INSERT** pro strukturu $\mathcal{P}_1$ v nejhorším případě odpovídá amortizované složitosti pro strukturu $\mathcal{P}$. Tedy můžeme shrnout

Věta 10.4. *Struktura $\mathcal{P}_1$ je semidynamická struktura řešící f. Algoritmus na vyčíslení f vyžaduje čas $O(Q_{\mathcal{S}}(n)\log(n))$, struktura $\mathcal{P}_1$ potřebuje $O(S_{\mathcal{S}}(n))$ paměti a časová složitost v nejhorším případě algoritmu implementujícího operaci* **INSERT** *je $O(\frac{P_{\mathcal{S}}(n)\log(n)}{n})$.*

Když $Q_{\mathcal{S}}(n) = n^{\varepsilon}$ pro nějaké $\varepsilon > 0$, pak algoritmus na vyčíslení f ve struktuře $\mathcal{P}$ vyžaduje čas $O(n^{\varepsilon})$.

Když $P_{\mathcal{S}}(n) = n^{\varepsilon}$ pro nějaké $\varepsilon > 1$, pak algoritmus pro operaci **INSERT** *ve struktuře $\mathcal{P}$ vyžaduje v nejhorším případě čas $O(n^{\varepsilon-1})$.*

Chceme-li přidat operaci **DELETE**, tak potřebujeme, aby pro $\mathcal{S}$-strukturu existovala operace, které se říká **Falešný DELETE** (algoritmus, který ji implementuje, budeme nazývat **Označení**(x, Z), to znamená vynechání x ve $\mathcal{S}$-struktuře reprezentující množinu Z). Když aplikujeme **Označení**(x) na strukturu $\mathcal{S}(A)$, nastane jedna z následujících dvou možností:

> $x \notin A$ a pak se struktura $\mathcal{S}(A)$ nezmění;
> $x \in A$ a pak se vytvoří struktura $\bar{\mathcal{S}}(A)$, která vyžaduje stejnou paměť jako $\mathcal{S}(A)$, a algoritmus, který pro vstup $y \in U_1$ spočítá $f(y, A \setminus \{x\})$ (místo $f(y, A)$) ve stejném čase jako algoritmus pro strukturu $\mathcal{S}(A)$ a vstup y.

Tuto operaci lze opakovat. Formálně, když na strukturu $\mathcal{S}(A)$ aplikujeme posloupnost operací **Označení**(x_1), **Označení**(x_2),..., **Označení**(x_k), pak výsledkem bude struktura $\bar{\mathcal{S}}(A \setminus \{x_i \mid i = 1,2,\ldots,k\})$, která vyžaduje stejnou paměť jako $\mathcal{S}(A)$, a algoritmus, který pro každý vstup $y \in U_1$ vyčíslí hodnotu $f(y, A \setminus \{x_i \mid i = 1,2,\ldots,k\})$. Protože tento postup nezmenšuje nároky na paměť (jsou $S_{\mathcal{S}}(|A|)$, nikoliv $S_{\mathcal{S}}(|A \setminus \{x_i \mid i = 1,2,\ldots,k\}|)$) ani časové nároky na výpočet f, ale počítá hodnotu f jako při **DELETE**, říká se mu **Falešný DELETE**. Například v datové struktuře uspořádané pole proběhne tak, že se prvek x pouze označí, ale následující prvky se neposouvají (odtud název **Označení**). Pro jednodušší komunikaci označíme $B = A \setminus \{x_i \mid i = 1,2,\ldots,k\}$ a $\bar{\mathcal{S}}(B)$ vzniklou strukturu a řekneme, že $\mathcal{S}(A)$ je <u>původcem</u> $\bar{\mathcal{S}}(B)$. Pak $\bar{\mathcal{S}}(B)$ vyžaduje $S_{\mathcal{S}}(|A|)$ paměti, pro $y \in U_1$ vypočteme $f(y, B)$ v čase $Q_{\mathcal{S}}(|A|)$ a pro $x \in U_2$ zkonstruujeme strukturu $\bar{\mathcal{S}}(B \setminus \{x\})$ v čase $D_{\mathcal{S}}(|A|)$. Budeme předpokládat, že funkce $D_{\mathcal{S}}$ je neklesající. Za těchto předpokladů následující konstrukce vytváří dynamickou strukturu pro vyčíslení f.

Konstrukce 10.5. Mějme dánu množinu $A \subseteq U_2$. Struktura $\mathcal{D}(A)$ se skládá ze souboru struktur $\{\bar{\mathcal{S}}(A_i) \mid i = 0,1,\ldots,4 + \lfloor \log(|A|-1) \rfloor\}$ a souboru spojových seznamů $\{s(A_i) \mid i = 0,1,\ldots,4 + \lfloor \log(|A|-1) \rfloor\}$ a z datové struktury T reprezentující množinu A (umožňující operace **MEMBER**(x), **INSERT**(x) a **DELETE**(x) v čase $O(\log|A|)$), a jsou splněné následující podmínky:

(d1) $A_i \cap A_j = \emptyset$ pro různá i,j a $A = \bigcup_i A_i$;

(d2) buď $A_i = \emptyset$ nebo $2^{i-3} < |A_i| \leq 2^i$ pro každé i;
(d3) když $A_i \neq \emptyset$, pak $\bar{\mathcal{S}}(A_i)$ je struktura pro množinu A_i, jejímž původcem je $\mathcal{S}(B_i)$, kde $|B_i| \leq 2^i$;
(d4) $s(A_i)$ je prostý seznam prvků z A_i a každý prvek v A_i je propojen oboustrannými ukazateli s jeho reprezentací ve struktuře T;
(d5) s každým prvkem v seznamu A_i je spojen index i a velikost množiny A_i.

Všimněte si, že když $A_i \neq \emptyset$, pak podle (d2) platí $2^{i-3} < |A|$, tedy $i-3 \leq 1+\lfloor\log(|A|-1)\rfloor$, a proto $i \leq 4 + \lfloor\log(|A|-1)\rfloor$. To vysvětluje podivné meze pro i.

Nyní popíšeme algoritmy pro $\mathcal{D}$-strukturu.

VYCIS(x)
zvol $i_0 \in \{0, 1, \ldots, 4 + \lfloor\log(|A|-1)\rfloor\}$ takové, že $A_{i_0} \neq \emptyset$
vypočti $f(x, A_{i_0})$, $f(x, A) := f(x, A_{i_0})$
for every $i = 0, 1, \ldots, 4 + \lfloor\log(|A|-1)\rfloor$ takové, že $A_i \neq \emptyset$ a $i \neq i_0$ **do**
  vypočti $f(x, A_i)$, $f(x, A) := f(x, A) \odot f(x, A_i)$
enddo

Korektnost algoritmu plyne z vlastností souboru množin $\{A_i \mid i = 0, 1, \ldots, 4 + \lfloor\log(|A|-1)\rfloor\}$ a struktury $\bar{\mathcal{S}}(A_i)$.

INSERT(x)
if $x \notin A$ **then**
  INSERT(x, T) (Komentář: vkládáme x do pomocné struktury T)
  najdi nejmenší j takové, že $1 + \sum_{i=0}^{j} |A_i| \leq 2^j$
  $s(A_j) := s(A_j) \cup \{x\}$
  ukazateli spoj x v seznamu $s(A_j)$ s x ve struktuře T
  for every $i < j$ **do**
    $s(A_j)$:=konkatenace $s(A_j)$ a $s(A_i)$, zruš $s(A_i)$ a $\mathcal{S}(A_i)$ (Komentář: $A_i = \emptyset$)
  enddo
  vypočti délku seznamu $s(A_j)$ (tj. velikost množiny $A_j = \{x\} \cup \bigcup_{i=0}^{j} A_i$)
  pomocí seznamu $s(A_j)$ vytvoř strukturu $\mathcal{S}(A_j)$
  přitom oprav u prvků v tomto seznamu odkazy na jméno množiny
endif

DELETE(x)
if x je reprezentován v T **then**
  najdi i takové, že x je v seznamu $s(A_i)$
  odstraň x ze seznamu $s(A_i)$
(Komentář: je splněna právě jedna z následujících podmínek a podle toho se provede akce)
  case
    if $|A_i| = 1$ **do** zruš $\mathcal{S}(A_i)$ a $s(A_i)$; (Komentář: $A_i = \emptyset$)
    if $\max\{2, 2^{i-3}+2\} \leq |A_i|$ **do Označení**$(x, \mathcal{S}(A_i))$;
    if $1 < 2^{i-3}+1 = |A_i|$ a ($A_{i-1} = \emptyset$ nebo $|A_{i-1}| \geq 2^{i-2}+1$) **do**
      vyměň $s(A_{i-1})$ a $s(A_i)$, $\mathcal{S}(A_i) := \mathcal{S}(A_{i-1})$
      průchodem $s(A_{i-1})$ vytvoř novou $\mathcal{S}$-strukturu $\mathcal{S}(A_{i-1})$;
    if $1 < 2^{i-3}+1 = |A_i|$ a $0 < |A_{i-1}| \leq 2^{i-2}$ **do**

$s(B) :=$ konkatenace $s(A_{i-1})$ a $s(A_i)$
zruš seznamy $s(A_{i-1})$ a $s(A_i)$ a struktury $\mathcal{S}(A_{i-1})$ a $\mathcal{S}(A_i)$
průchodem seznamu $s(B)$ vytvoř $\mathcal{S}$-strukturu $\mathcal{S}(B)$
if $|B| < 2^{i-2}$ **then**
$\mathcal{S}(A_{i-1}) := \mathcal{S}(B)$, $s(A_{i-1}) := s(B)$ (Komentář: $A_i = \emptyset$)
else
$\mathcal{S}(A_i) := \mathcal{S}(B)$, $s(A_i) := s(B)$ (Komentář: $A_{i-1} = \emptyset$)
endif
endcase
DELETE(T, x)
(Komentář: odstraníme x z pomocné struktury T)
endif

Ověření korektnosti algoritmů **INSERT** a **DELETE** je přímočaré (v zápise algoritmů jsme pro jednoduchost vynechali označování struktur pruhy).

Poznámka. Všimněme si, že když **INSERT** vytváří $\mathcal{S}$-strukturu $\mathcal{S}(A_i)$, pak $2^{i-1} < |A_i| \leq 2^i$, když **DELETE** vytváří $\mathcal{S}$-strukturu $\mathcal{S}(A_i)$, pak $2^{i-2} \leq |A_i| \leq 2^{i-1}$.

Abychom spočítali amortizovanou složitost operací, použijeme stejně jako pro semidynamizaci metodu potenciálu. Zde však potenciálová funkce bude komplikovanější. Pro jednodušší zápis označme $\lambda(n) = 1 + \lfloor \log_2 n \rfloor$ a $\mu(n) = 4 + \lfloor \log_2(n-1) \rfloor$. Pak binární zápis přirozeného čísla n má právě $\lambda(n)$ cifer. Předpokládejme, že jsme pomocí struktury $\mathcal{D}$ reprezentovali množinu A a nechť A_i pro $i = 0, 1, \ldots, \mu(|A|)$ jsou podmnožiny z definice struktury $\mathcal{D}$. Z definice $\mathcal{D}$-struktury víme, že když $A_j \neq \emptyset$, pak $j \leq \mu(|A|)$ a existuje j takové, že $\lambda(|A|) \leq j$ a $A_j \neq \emptyset$. Označme $\nu(\mathcal{D}(A)) = \max\{\max\{j \mid A_j \neq \emptyset\}, 1 + \lambda(|A|)\}$. Pro přirozené číslo i definujme funkce ϕ_i^I a ϕ_i^D předpisem

$$\phi_i^I(A) = \frac{\sum_{k=0}^{i-1} |A_k|}{2^i} P(2^i) \quad \text{a} \qquad \phi_i^D(A) = \begin{cases} 0 & \text{když } i \leq 3 \text{ nebo } A_i = \emptyset, \\ \frac{2^i - |A_i|}{2^i - 2^{i-3}} P(2^i) & \text{když } i > 3 \text{ a } A_i \neq \emptyset. \end{cases}$$

a pak potenciál je funkce

$$\phi(A) = \sum_{i=0}^{\nu(\mathcal{D}(A))} (2\phi_i^I(A) + \phi_i^D).$$

V odhadu amortizované složitosti budeme využívat, že funkce $\frac{P_{\mathcal{S}}(n)}{n}$ je neklesající, a proto pro $n \leq m$ platí

$$P_{\mathcal{S}}(n) \leq \frac{n}{m} P_{\mathcal{S}}(m) \leq P_{\mathcal{S}}(m).$$

Nejprve odhadneme amortizovanou složitost pro operaci **INSERT**(x). Když aplikujeme operaci **INSERT**(x), tak nejdřív pomocí struktury T testujeme, zda $x \in A$. Když $x \in A$, pak operace končí a $\mathcal{D}$-struktura reprezentující A se nemění a ani potenciál se nemění. Pak amortizovaná složitost operace **INSERT**(x) je stejná jako její časová složitost a z předpodkladů na strukturu T je to $O(\log |A|)$. Když $x \notin A$, pak operace nalezne nejmenší i takové, že $1 + \sum_{j=0}^{i} |A_j| \leq 2^j$. Pak nutně platí $i \leq \lambda(|A|)$. Operace vytvoří $\mathcal{D}$-strukturu pro množinu $B = A \cup \{x\}$ takovou, že $B_j = \emptyset$ pro $j < i$, $B_i = \{x\} \cup \bigcup_{j=0}^{i} A_j$ a $B_j = A_j$ pro

$j > i$. Pak upraví seznamy množin B_j (pomocí seznamů množin A_j), vytoří $\mathcal{S}$-strukturu pro množinu B_i, upraví strukturu T, aby reprezentovala množinu B a upraví ukazatele spojené s prvkem x. Na to operaci stačí čas $O(P_{\mathcal{S}}(2^i) + \log|A|)$.

Nyní musíme odhadnout $\phi(B) - \phi(A)$ a pak srovnat s časem potřebným pro vykonání operace. Tím dostaneme odhad amortizované složitosti operace **INSERT**(x) pro tento případ. Pro $j > i$ platí $A_j = B_j$, pro $j < i$ je $B_j = \emptyset$ a $B_i = \{x\} + \bigcup_{j=0}^{i} A_i$, tedy $|B_i| = 1 + \sum_{j=0}^{i} A_i$. Proto dostáváme, že pro $j > i$ platí $\sum_{k=0}^{j-1} |B_k| = 1 + \sum_{k=0}^{j-1} |A_k|$. Tedy pro j takové, že $i < j \leq \nu(\mathcal{D}(A))$ platí

$$\phi_j^I(B) - \phi_j^I(A) = \frac{P_{\mathcal{S}}(2^j)}{2^j} \quad \text{a} \quad \phi_j^D(B) - \phi_j^D(A) = 0.$$

Pro j takové, že $0 \leq j < i$ platí (v odhadu pro ϕ_j^D předpokládáme, že $j > 3$)

$$\phi_j^I(B) - \phi_j^I(A) = \frac{-\sum_{k=0}^{j-1} |A_k|}{2^j} P_{\mathcal{S}}(2^j) \leq 0, \qquad \phi_j^D(B) - \phi_j^D(A) = -\frac{2^j - |A_j|}{2^j - 2^{j-3}} P_{\mathcal{S}}(2^j) \leq 0.$$

Odtud pro $j \in \{0, 1, \ldots, \nu(\mathcal{D}(A))\}$ takové, že $j \neq i$, dostáváme

$$2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A)) \leq 2\frac{P_{\mathcal{S}}(2^j)}{2^j}.$$

Nakonec pro i dostáváme vztahy

$$\phi_i^I(B) - \phi_i^I(A) = \frac{-\sum_{k=0}^{i-1} |A_k|}{2^i} P_{\mathcal{S}}(2^i) \leq -\frac{P_{\mathcal{S}}(2^i)}{2}$$

$$\phi_i^D(B) - \phi_i^D(A) = \begin{cases} -\frac{1+\sum_{k=0}^{i-1} |A_k|}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) \leq -\frac{2^{i-1} P_{\mathcal{S}}(2^i)}{2^i - 2^{i-3}} & \text{když } A_i \neq \emptyset \\ \frac{2^i - 1 - \sum_{k=0}^{i-1} |A_k|}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) \leq \frac{2^{i-1} P_{\mathcal{S}}(2^i)}{2^i - 2^{i-3}} & \text{když } A_i = \emptyset, \end{cases}$$

protože $1 + \sum_{k=0}^{i-1} > 2^{i-1}$. Protože $\frac{2^{i-1}}{2^i - 2^{i-3}} = \frac{4}{7}$, tak dostaneme

$$2(\phi_i^I(B) - \phi_i^I(A)) + (\phi_i^D(B) - \phi_i^D(A)) \leq -\frac{3}{7} P_{\mathcal{S}}(2^i).$$

Protože $\phi(A) = \sum_{i=0}^{\nu(\mathcal{D}(A))} (2\phi_i^I(A) + \phi_i^D(A))$ tak dostáváme

$$\phi(B) - \phi(A) = \sum_{j=0}^{\nu(\mathcal{D}(A))} \left(2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A))\right) \leq$$

$$\left(\sum_{j=0}^{\nu(\mathcal{D}(A))} \frac{P_{\mathcal{S}}(2^j)}{2^j} \right) - \frac{3}{7} P_{\mathcal{S}}(2^i) \leq$$

$$\frac{P_{\mathcal{S}}(8|A|)}{8|A|}(4 + \log|A|) - \frac{3}{7} P_{\mathcal{S}} 2^i.$$

Protože můžeme volit časovou jednotku, tak daná funkce určuje spotřeby času až na multiplikativní konstantu. Proto můžeme předpokládat, že konstrukce $\mathcal{S}$-struktury pro n-prvkovou množinu spotřebuje nejvýše $\frac{3}{7} P_{\mathcal{S}}(n)$ času a tedy dostáváme

Lemma 10.6. *Amortizovaný čas operace* **INSERT**(x) *pro reprezentovanou množinu* A*, je* $O(\frac{P_{\mathcal{S}}(8|A|)}{|A|}\log|A|)$. □

Nyní odhadneme amortizovanou složitost operace **DELETE**(x). Operace začne stejně jako operace **INSERT**(x) testem, zda $x \in A$, ke kterému použije strukturu T. Když $x \notin A$, tak operace končí a protože se struktura nemění, tak ani její potenciál se nemění. Tedy amortizovaná složitost je stejná jako časová složitost a můžeme předpokádat, že je $O(\log|A|)$. Když $x \in A$, pak nalezne i takové, že $x \in A_i$ zjistí velikosti $|A_i|$ a $|A_{i-1}|$, tyto údaje nalezne pomocí ukazatelů v čase $O(1)$. Když $|A_i| = 1$, pak smaže A_i a údaje s tím spojené (vyžaduje to čas $O(\log|A|)$). Když $A_i > 2^{i-3}+1$ (a zároveň $|A_i| \neq 1$), pak zavolá pomocnou proceduru **Označení**(x, A_i), odstraní x ze struktury T a také ukazatele spojené s x a upraví údaje o množině A_i. To vyžaduje čas $O(\log|A|+D_{\mathcal{S}}(A_i)) = O(\log|A|+D_{\mathcal{S}}(|A|))$. V obou případech se $\mathcal{D}$-struktura se změní tak, že $B_i = A_i \setminus \{x\}$ a $B_j = A_j$ pro $j \neq i$. To znamená, že když $j \leq i$, pak $\phi_j^I(B) = \phi_j^I(A)$, a když $j > i$, pak $\phi_j^I(B) - \phi_j^I(A) = -\frac{P_{\mathcal{S}}(2^j)}{2^j} \leq 0$. Dále když $j \neq i$, pak $\phi_j^D(B) = \phi_j^D(A)$ a nakonec $\phi_i^D(B) - \phi_i^D(A) = \frac{P_{\mathcal{S}}(2^i)}{2^i - 2^{i-3}} = O(\frac{P_{\mathcal{S}}(|A|)}{|A|})$, protože $\frac{7}{8}\frac{n}{2^i-2^{i-3}} = \frac{7}{8}\frac{n}{(2^3-1)2^{i-3}} = \frac{7}{8}\frac{n}{7(2^{i-3})} = \frac{n}{2^i}$ pro každé číslo n a přirozené číslo i. Tedy amortizovanou složitost v tomto případě lze odhadnout výrazem $O(\log|A| + D_{\mathcal{S}}(|A|) + \frac{P_{\mathcal{S}}(|A|)}{|A|})$.

Zbývá poslední případ, když $|A_i| = 2^{i-3}+1$. Když $|A_{i-3}| \leq 2^{i-3}$, pak $B_{i-1} = (A_i \cup A_{i-1}) \setminus \{x\}$, $B_i = \emptyset$ and $B_j = A_j$ pro $j \neq i, i-1$. Algoritmus vytvoří $\mathcal{S}$-strukturu pro B_{i-1}, aktualizuje velikosti množin B_i a B_{i-1} a upraví ukazatele a hodnoty prvků v B_{i-1}. Když $2^{i-3} < |A_{i-1}| \leq 2^{i-2}$, pak $B_i = (A_i \cup A_{i-1}) \setminus \{x\}$, $B_{i-1} = \emptyset$ a $B_j = A_j$ pro $j \neq i, i-1$. Algoritmus vytvoří $\mathcal{S}$-strukturu pro B_i, aktualizuje velikosti množin B_i a B_{i-1} a upraví ukazatele a hodnoty prvků v B_i. V těchto případech operace vyžaduje čas

$$O(\log|A| + P_{\mathcal{S}}(|A_i| + |A_{i-1}| - 1)) = O(\log|A| + P_{\mathcal{S}}(3(2^{i-3})).$$

Když $|A_{i-1}| > 2^{i-2}$, pak $B_{i-1} = A_i \setminus \{x\}$, $B_i = A_{i-1}$ a $B_j = A_j$ pro $j \neq i, i-1$. Algoritmus vytvoří $\mathcal{S}$-strukturu pro B_{i-1}, aktualizuje velikosti množin B_i a B_{i-1} a upraví ukazatele a hodnoty prvků v B_{i-1} a B_i. To vyžaduje čas $O(\log|A| + P_{\mathcal{S}}(2^{i-3}))$.

Nyní opět odhadneme rozdíl $\phi(B) - \phi(A)$ a srovnáním s časem potřebným pro vykonání operace dostaneme odhad amortizované složitosti operace. Víme, že $A_j = B_j$ pro $j \neq i, i+1$ a $|A_{i-1}| + |A_i| = |B_{i-1}| + |B_i| + 1$, proto $\phi_j^I(B) = \phi_j^I(A)$ pro $j < i$ a $\phi_j^I(B) - \phi_j^I(A) = -\frac{P_{\mathcal{S}}(2^j)}{2^j - 2^{j-3}} \leq 0$ pro $j > i$ a $\phi_j^D(B) = \phi_j^D(A)$ pro $j \neq i-1, i$. Další postup rozdělíme do čtyř případů – $|A_{i-1}| = 0$, $0 \neq |A_{i-1}| \leq 2^{i-3}$, $2^{i-3} < |A_{i-1}| \leq 2^{i-2}$ a $2^{i-2} < |A_{i-1}|$.

(i) $|A_{i-1}| = 0$. Nyní $B_{i-1} = A_i \setminus \{x\}$ a $B_i = \emptyset$ a tedy platí

$$\begin{aligned}\phi_i^I(B) - \phi_i^I(A) &= \frac{2^{i-3}}{2^i} P_{\mathcal{S}}(2^i) = \frac{P_{\mathcal{S}}(2^i)}{8},\\ \phi_{i-1}^D(B) - \phi_{i-1}^D(A) &= \frac{2^{i-1} - 2^{i-3}}{2^{i-1} - 2^{i-4}} P_{\mathcal{S}}(2^{i-1}) = \frac{6}{7} P_{\mathcal{S}}(2^{i-1}) \leq \frac{3}{7} P_{\mathcal{S}}(2^i),\end{aligned}$$

(kde jsme využili monotonii funkce $\frac{P_{\mathcal{S}}(n)}{n}$)

$$\phi_i^D(B) - \phi_i^D(A) = -\frac{2^i - 2^{i-3} - 1}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) \leq -\frac{6}{7} P_{\mathcal{S}}(2^i),$$

kde jsme použili, že pro $i \geq 3$ platí $\frac{1}{2^i - 2^{i-3}} - 1 \leq -\frac{6}{7}$. Tedy můžeme shrnout dosažené odhady

$$\phi(B) - \phi(A) = \sum_{j=0}^{\nu(\mathcal{D}(A))} \big(2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A))\big) \leq \frac{7 + 12 - 24}{28} P_{\mathcal{S}}(2^i) = -\frac{5}{28} P_{\mathcal{S}}(2^i).$$

(ii) $0 \neq |A_{i-1}| \leq 2^{i-3}$. Pak $B_{i-1} = (A_{i-1} \cup A_i) \setminus \{x\}$, $B_i = \emptyset$ a tedy

$$\begin{aligned}
\phi_i^I(B) - \phi_i^I(A) =& \frac{2^{i-3}}{2^i} P_{\mathcal{S}}(2^i) = \frac{P_{\mathcal{S}}(2^i)}{8}, \\
\phi_{i-1}^D(B) - \phi_{i-1}^D(A) =& -\frac{2^{i-3}}{2^{i-1} - 2^{i-4}} P_{\mathcal{S}}(2^{i-1}) = -\frac{2}{7} P_{\mathcal{S}}(2^{i-1}) \leq 0, \\
\phi_i^D(B) - \phi_i^D(A) =& -\frac{2^i - 2^{i-3} - 1}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) \leq -\frac{6}{7} P_{\mathcal{S}}(2^i).
\end{aligned}$$

Takže dostáváme

$$\phi(B) - \phi(A) = \sum_{j=0}^{\nu(\mathcal{D}(A))} \big(2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A))\big) \leq \frac{7 - 24}{28} P_{\mathcal{S}}(2^i) = -\frac{17}{28} P_{\mathcal{S}}(2^i).$$

(iii) $2^{i-3} < |A_{i-1}| \leq 2^{i-2}$. Pak $B_{i-1} = \emptyset$ a $B_i = (A_{i-1} \cup A_i) \setminus \{x\}$ a dostaneme

$$\begin{aligned}
\phi_i^I(B) - \phi_i^I(A) =& -\frac{|A_{i-1}|}{2^i} P_{\mathcal{S}}(2^i) \leq -\frac{P_{\mathcal{S}}(2^i)}{8}, \\
\phi_{i-1}^D(B) - \phi_{i-1}^D(A) =& \frac{|A_{i-1}| - 2^{i-1}}{2^{i-1} - 2^{i-4}} P_{\mathcal{S}}(2^{i-1}) \leq \frac{-2^{i-2}}{2^{i-1} - 2^{i-4}} P_{\mathcal{S}}(2^{i-1}) = -\frac{4}{7} P_{\mathcal{S}}(2^{i-1}), \\
\phi_i^D(B) - \phi_i^D(A) =& \frac{-|A_{i-1}|}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) \leq -\frac{2^{i-3}}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i) = -\frac{P_{\mathcal{S}}(2^i)}{7},
\end{aligned}$$

Shrnutím těchto odhadů dostáváme

$$\phi(B) - \phi(A) = \sum_{j=0}^{\nu(\mathcal{D}(A))} \big(2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A))\big) \leq \frac{-7 - 4}{28} P(2^i) = -\frac{11}{28} P(2^i).$$

(iv) $2^{i-2} < |A_{i-1}|$. Protože $B_{i-1} = A_i \setminus \{x\}$ a $B_i = A_{i-1}$, tak platí

$$\begin{aligned}
\phi_i^I(B) - \phi_i^I(A) =& \frac{|A_i| - 1 - |A_{i-1}|}{2^i} P_{\mathcal{S}}(2^i), \\
\phi_{i-1}^D(B) - \phi_{i-1}^D(A) =& -\frac{|A_i| - 1 - |A_{i-1}|}{2^{i-1} - 2^{i-4}} P_{\mathcal{S}}(2^{i-1}) \leq \frac{|A_{i-1}| - |A_i| + 1}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i), \\
\phi_i^D(B) - \phi_i^D(A) =& \frac{|A_i| - |A_{i-1}|}{2^i - 2^{i-3}} P_{\mathcal{S}}(2^i),
\end{aligned}$$

kde jsme využili, že $|A_{i-1}| > 2^{i-2} > 2^{i-3} = |A_i| - 1 = |B_{i-1}|$ a $P_{\mathcal{S}}(2^{i-1}) \leq \frac{P_{\mathcal{S}}2^i}{2}$. Shrneme

$$\begin{aligned}\phi(B) - \phi(A) &= \sum_{j=0}^{\nu(\mathcal{D}(A))} \big(2(\phi_j^I(B) - \phi_j^I(A)) + (\phi_j^D(B) - \phi_j^D(A))\big) \\ &\leq \frac{|A_i| - |A_{i-1}|}{2^i} P_{\mathcal{S}}(2^i) \leq \frac{2^{i-3} - 2^{i-2}}{2^i} P_{\mathcal{S}}(2^i) = \frac{-P_{\mathcal{S}}(2^i)}{8}.\end{aligned}$$

Tedy existuje $c > 0$ takové, že všech čtyřech případech $\phi(B) - \phi(A) \geq -cP_{\mathcal{S}}(2^i)$. Když použijeme stejný argument o volbě časové jednotky jako pro operaci **INSERT**, tak dostaneme

Lemma 10.7. *Amortizovaný čas operace* **DELETE**(x) *pro reprezentovanou množinu A, je $O(\log|A| + D_{\mathcal{S}}(8|A|) + \frac{P_{\mathcal{S}}(8|A|)}{|A|})$.* □

Neformálně řečeno, když operace **DELETE** vytváří $\mathcal{S}$-strukturu pro A_i, pak aby se znovu vytvářela $\mathcal{S}$-struktura pro A_i musíme označit aspoň 2^{i-3} prvků v množině A_i neboli vytváříme novou $\mathcal{S}$-strukturu pro A_i nejvýše jednou během 2^{i-3} úspěšných **DELETE** v množině A_i. Bohužel stejný neformální argument nelze přímo použít pro operaci **INSERT**, protože při vytváření $\mathcal{S}$-struktury pro A_i se mísejí operace **INSERT** a **DELETE**. Proto je potenciálová funkce pro dynamickou verzi tak složitá. Takže shrneme dosažené výsledky

Věta 10.8. *Struktura $\mathcal{D}$ je dynamická struktura řešící problém f pro množinu o velikosti n, která na výpočet f vyžaduje čas $O(Q_{\mathcal{S}}(8n)\log n)$ a potřebuje $O(S_{\mathcal{S}}(8n))$ paměti. Amortizovaná složitost operace* **INSERT** *je $O(\frac{P_{\mathcal{S}}(n)\log n}{n})$ a operace* **DELETE** *je $O(D_{\mathcal{S}}(n) + \log n + \frac{P_{\mathcal{S}}(8n)}{n})$.*

Když funkce $Q_{\mathcal{S}}(n)$ (respektive $S_{\mathcal{S}}(n)$) je omezena polynomem, pak výpočet f vyžaduje čas $O(Q_{\mathcal{S}}(n)\log n)$ (respektive $\mathcal{D}$-struktura vyžaduje $O(S_{\mathcal{S}}(n)$ paměti).

Když $Q_{\mathcal{S}}(n) = n^\varepsilon$ pro nějaké $\varepsilon > 0$, pak vyčíslení f vyžaduje čas $O(n^\varepsilon)$.

Když $P(n) = n^\varepsilon$ pro nějaké $\varepsilon > 1$, pak amortizovaná složitost operace **INSERT** *je $O(n^{\varepsilon-1})$ a operace* **DELETE** *je $O(D(n) + n^{\varepsilon-1})$.*

Důkaz. Stačí si uvědomit, že $2^{3+\log n} = 8n$ a že když $Q_{\mathcal{S}}(n)$ (respektive $S_{\mathcal{S}}(n)$) je superpolynomiální, pak $Q_{\mathcal{S}}(8n) \not\leq cQ_{\mathcal{S}}(n)$ (respektive $S_{\mathcal{S}}(8n) \not\leq cS_{\mathcal{S}}(n)$) pro každou konstantu c. Zbytek je přímá aplikace předchozích argumentů. □

Věta 10.8 má dvě významná zobecnění, jejichž důkaz je technicky náročný a přesahuje rámec této přednášky. První zobecnění říká, že i metodu dynamizace lze modifikovat tak, že časová složitost v nejhorším případě je stejná jako amortizovaná složitost ve Větě 10.8. Zde je problém, jak rozdělit vytváření $\mathcal{S}$-struktur a přitom umožnit výpočet f (a zbytečně nemíchat operace **INSERT** a **DELETE** dohromady). Druhé zobecnění se týká vztahu složitostí jednotlivých operací a ukazuje, že zrychlení aktualizačních operací je kompenzováno zhoršením složitosti vyčíslení f a naopak. Tyto konstrukce jsou založeny na různých rozvojích přirozených čísel (důkazy zde prezentovaných vět používaly binární rozvoj). Lze použít libovolný číselný rozvoj přirozených čísel, ale znalosti rozvoje přirozených čísel vyžadují podrobnější znalosti z teorie čísel.

Věta 10.9. *Mějme rozložitelný vyhledávací problém f a nechť $\mathcal{S}$ je statická struktura řešící f, která umožňuje* **Falešný DELETE**. *Pak pro hladkou funkci ϕ (má všechny derivace, které jsou spojité) existuje dynamická struktura $\mathcal{D}$ řešící f taková, že*

vyčíslení f v nejhorším případě vyžaduje čas $O(\phi(n)Q_{\mathcal{S}}(n))$;
operace **DELETE** *v nejhorším případě vyžaduje čas $O(\log n + D_{\mathcal{S}}(n) + \frac{P_{\mathcal{S}}(n)}{n})$;*
operace **INSERT** *v nejhorším případě vyžaduje čas $O(\frac{\log n}{\log(\frac{\phi(n)}{\log n})}\frac{P_{\mathcal{S}}(n)}{n})$, když $\phi(n) = \Omega(\log(n))$, a $O(\phi(n)n^{\frac{1}{\phi(n)}}\frac{P_{\mathcal{S}}(n)}{n})$, když $\phi(n) = O(\log n)$.*

Důkaz neuvádíme.

Metody dynamizace zavedl Bentley 1979. Větu o semidynamizaci dokázali Bentley a Saxe 1980. Věty o dynamizaci dokázali Overmars a Leeuwen 1981, zobecnění pro jiné rozvoje ukázali Mehlhorn a Overmars 1981. Použitý výpočet amortizované složitosti navrhnul student Martin Babka.

11. Problém **UNION-FIND**

Tato část se zabývá jednou speciální úlohou, která se vyskytuje v mnoha problémech. Nejprve popíšeme její zadání.

Úloha **UNION-FIND**.
Je dán rozklad $\mathcal{S}$ univerza $U = \{1, 2, \ldots, n\}$ o velikosti n (předpokládáme, že v počítači lze pole o velikosti n realizovat). Připomínáme, že rozklad je soubor neprázdných navzájem disjunktních množin, jejichž sjednocením je univerzum U. Každá množina v rozkladu má své jméno. Na počátku úlohy jsou všechny množiny v $\mathcal{S}$ jednoprvkové. Naším úkolem je provést posloupnost $\mathfrak{P}$ následujících dvou operací

UNION(A, B, C)
kde $A, B \in \mathcal{S}$, výsledkem operace je odstranění množin A a B z rozkladu $\mathcal{S}$ a jejich nahrazení množinou $C = A \cup B$;
FIND(x)
kde $x \in U$, výsledkem operace je jméno množiny $A \in \mathcal{S}$ takové, že $x \in A$.

Tato úloha je podproblémem např. při práci s EQUIVALENCE and COMMON tvrzeními ve Fortranu, v algoritmu konstruujícím minimální kostru grafu, v algoritmu počítajícím dominátor v orientovaném grafu, při kontrole flow-grafů při reducibilitě, v algoritmech na výpočet hloubky ve stromech nebo při hledání následníků ve stromech, při konstrukci redukovaného automatu, v algoritmech řešících minimální problémy, atd. To ukazuje na význam této úlohy, a proto byla jejímu řešení věnována velká pozornost. Existují dvě hlavní strategie řešení podle toho, kterou operaci preferují. Nejprve popíšeme struktury preferující operaci **FIND**.

Nejjednoduší datová struktura pro tuto úlohu je pole $R[1, ..., n]$ takové, že pro $x \in U$ je $R(x) = A$, právě když $x \in A \in \mathcal{S}$. Pak operace **FIND** vyžaduje čas $O(1)$, ale čas operace **UNION** je $O(n)$ (musíme projít všechny prvky $x \in U$ a pokud $R(x) \in \{A, B\}$, změnit $R(x)$ na C). Když je ale navíc pro každou množinu $A \in \mathcal{S}$ dán prostý spojový seznam $s(A)$ jejích prvků, pak operace **UNION** provede konkatenaci spojových seznamů $s(A)$ a $s(B)$, výsledný seznam nazve $s(C)$ a změní hodnotu $R(x)$ jen u prvků v seznamu $s(C)$. To vyžaduje čas $O(|A| + |B|)$.

Avšak podstatným vylepšením je až rozlišení vnitřní a vnější reprezentace názvů množin. Pak stačí změnit hodnoty pole R jen u prvků z menší množiny. Tato reprezentace kromě pole R a seznamů $s(A)$ pro každé $A \in \mathcal{S}$ má další tři pole: MAPIN, MAPOUT a SIZE. Pole MAPIN přiřazuje názvu množiny jeho vnitřní reprezentaci, pole SIZE udává její velikost a MAPOUT pro vnitřní kód množiny uchovává její skutečný (vnější) název. Algoritmy mají tvar:

FIND(x)
$i := R(x)$
Výstup: $MAPOUT(i)$

UNION(A, B, C)
if $SIZE(A) < SIZE(B)$ **then**
$i := MAPIN(A)$, $j := MAPIN(B)$, $X := A$
else
$i := MAPIN(B)$, $j := MAPIN(A)$, $X := B$
endif
for every $x \in s(X)$ **do** $R(x) := j$ **enddo**
$SIZE(C) := SIZE(A) + SIZE(B)$
v polích MAPIN a SIZE zruš položky pro A a B a vytvoř položku pro C
$MAPIN(C) := j$, $MAPOUT(j) := C$, $MAPOUT(i) := NIL$

Pak platí

Věta 11.1. *Popsaná datová struktura realizuje posloupnost m operací* **FIND** *a $n-1$ operací* **UNION** *v čase $O(m + n \log n)$.*

Důkaz. Je zřejmé, že operace **FIND** vyžaduje čas $O(1)$. Tedy m operací **FIND** vyžaduje čas $O(m)$. Pro prvek $x \in U$ nechť $p(x)$ udává, kolikrát byla změněna hodnota $R(x)$. Po provedení celé posloupnosti operací je celkový čas spotřebovaný operacemi **UNION** $n - 1 + \sum_{x \in U} p(x)$. Když ukážeme, že pro každé $x \in U$ platí $p(x) \leq \log_2 n$, pak čas vyžadovaný všemi operacemi **UNION** je $O(n \log n)$ a věta bude dokázána.

Nejprve dokážeme, že platí následující invariant:

pro každý prvek $x \in U$ platí, že když $x \in A$ a $p(x) = i$, pak $|A| \geq 2^i$.

Na začátku každý prvek $x \in U$ leží v jednoprvkové množině a $p(x) = 0$, tedy invariant je splněn. Předpokládejme, že invariant je splněn po k operacích. Když $(k + 1)$-ní operace je **FIND**, pak tato operace nemění hodnoty $R(x)$ pro žádné $x \in U$ ani množiny v rozkladu $\mathcal{S}$. Proto invariant po této operaci platí. Když $(k + 1)$-ní operace je **UNION**(A, B, C), můžeme bez újmy na obecnosti předpokládat, že $|A| \leq |B|$. Pak pro každé $x \in A$ se změní hodnota $R(x)$, a proto vzroste $p(x)$ o 1, a pro $x \in U \setminus A$ se $p(x)$ nezmění. Protože pro každé $x \in U$ se velikost množiny obsahující x může jenom zvětšit, tak invariant zřejmě platí pro $x \in U \setminus A$. Dále $|A| \leq |B|$ implikuje, že $|C| \geq 2|A|$. Když pro $x \in A$ bylo před provedením operace $p(x) = i$, tak po jejím provedení je $p(x) = i + 1$ a platí $|C| \geq 2|A| \geq 22^i = 2^{i+1}$, a tedy invariant je opět splněn. Protože pro rozklad $\mathcal{S}$ vždy platí, že $|A| \leq n$ pro každou množinu $A \in \mathcal{S}$, dostáváme, že $p(x) \leq \log_2 n$ pro každé $x \in U$, a tvrzení je dokázáno. □

Nyní popíšeme stručně vývoj druhé struktury preferující operaci **UNION**.

V této struktuře je každá množina $A \in \mathcal{S}$ reprezentována stromem, takže je zde bijekce mezi vrcholy stromu a prvky reprezentované množiny. Strom je reprezentován tak, že každý vrchol má ukazatel na svého otce (jeho hodnota pro kořen je NIL). Navíc v kořeni stromu je uvedeno jméno množiny, případně i její velikost.

Pro každý prvek $x \in U$ je dán ukazatel vrchol(x) na ten vrchol stromu, který reprezentuje x. Realizace operace **FIND**(x) použije tento ukazatel, pak pomocí ukazatelů na otce vyhledá kořen a tam nalezne požadované jméno množiny. Operace **UNION**(A, B, C) se provede tak, že kořen stromu reprezentujícího jednu množinu se stane synem kořene stromu reprezentujícího druhou množinu a jméno množiny v kořeni výsledného stromu se přepíše na C. Jak bylo vidět v předchozí reprezentaci, je výhodné, aby se kořen menšího stromu stal synem kořene většího stromu. Abychom tohoto pozorování mohli využít, budeme uchovávat a aktualizovat i velikost reprezentované množiny. Popíšeme formálně algoritmy pro tyto operace.

UNION(A, B, C)
if size(A) < size(B) **then**
  otec(kořen A) := kořen B, nazev(B) := C, size(C) := size(B) + size(A)
  zruš nazev(A), size(A) a size(B)
else
  otec(kořen B) := kořen A, nazev(A) := C, size(C) := size(B) + size(A)
  zruš nazev(B), size(A) a size(B)
endif

FIND(x)
v := vrchol(x)
while otec(v) $\neq NIL$ **do** v := otec(v) **enddo**
Výstup: nazev(v)

Nejprve dokážeme složitost algoritmů v této struktuře a pak navrhneme její vylepšení.

Věta 11.2. *Když se posloupnost operací obsahující m operací* **FIND** *a $n-1$ operací* **UNION** *realizuje ve struktuře preferující operaci* **UNION**, *pak tato její realizace vyžaduje čas $O(n + m \log n)$.*

Důkaz. Je zřejmé, že operace **UNION** vyžaduje čas $O(1)$ a operace **FIND**(x) vyžaduje čas $O(h(\text{vrchol}(x))$, kde $h(x)$ je hloubka vrcholu x. Musíme tedy najít odhad hloubky vrcholů v této struktuře. Odvození odhadu je jednoduchou modifikací důkazu předchozí věty. Dokážeme, že platí následující invariant:

> pro každý vrchol $x \in U$ platí, že když hloubka vrchol(x) je i, pak $|A| \geq 2^i$, kde $x \in A \in \mathcal{S}$.

Na počátku jsou všechny prvky v jednoprvkových množinách a k nim příslušné vrcholy mají hloubku 0. Tedy invariant je splněn na počátku běhu algoritmu. Předpokládejme, že invariant platí po provedení k-té operace. Protože operace **FIND** nemění ani stromy ani množiny, tak pokud $(k+1)$-ní operace je **FIND**, tak invariant platí i po této operaci. Nechť nyní $(k+1)$-ní operace je **UNION**(A, B, C) a bez újmy na obecnosti předpokládejme, že $|A| \leq |B|$. Pak pro $x \in A$ se hloubka vrchol(x) zvětší o 1 a pro prvky $x \in U \setminus A$ se hloubka

vrchol(x) nezmění. Dále pro každý prvek $x \in U$ se velikost množiny v $\mathcal{S}$ obsahující x může jen zvětšit, a proto po provedení operace **UNION**(A, B, C) invariant platí pro všechna $x \in U \setminus A$. Uvažujme, že $x \in A$, a předpokládejme, že hloubka vrchol(x) před provedením operace je i. Protože $|A| \leq |B|$, tak z popisu operace a z indukčního předpokladu plyne $|C| = |A| + |B| \geq 2|A| \geq 22^i = 2^{i+1}$ a invariant je opět splněn. Z invariantu dále plyne, že hloubka každého vrcholu je nejvýše $\log_2 n$. Protože čas spotřebovaný operací **FIND**(x) je úměrný hloubce vrchol(x), dostáváme, že operace **FIND** spotřebuje $O(\log n)$ času, a tedy m operací **FIND** spotřebuje $O(m \log n)$ času a věta je dokázána. □

Přirozený způsob, jak zeefektivnit algoritmus pro operaci **FIND**(x), je modifikovat tento algoritmus tak, že všechny vrcholy na cestě z vrchol(x) do kořene se stanou syny kořene stromu. Pak některá operace **FIND**(x) sice vyžaduje více času (zvětší se multiplikativní konstanta), ale důsledkem je, že řada následujících operací **FIND** bude naopak rychlejší (když vrchol(y) bude synem kořene stromu, pak operace **FIND**(y) bude vyžadovat jen $O(1)$ času). Formální popis algoritmu **FIND** je následující:

FIND(x)
$v :=$ vrchol(x), $V := \{v\}$
while otec$(v) \neq NIL$ **do** $V := V \cup \{v\}$, $v :=$ otec(v) **enddo**
for every $u \in V$ **do** otec$(u) := v$ **enddo**
Výstup: nazev(v)

Naším cílem bude ukázat, že tato modifikace je efektivní a poskytuje prakticky lineární algoritmus. Pro tento účel zadefinujme jednu verzi Ackermannovy funkce A a její pseudoinverzi α.

Definujme

$$A(i,0) = 0, \quad A(i,1) = 2 \qquad \text{pro všechna } i \geq 0,$$
$$A(0,x) = 2x \qquad \text{pro všechna } x > 0,$$
$$A(i+1, x+1) = A(i, A(i+1,x)) \qquad \text{pro všechna } i, x \geq 0,$$
$$\alpha(m,n) = \min\{z \geq 1 \mid A(z, 4\lceil \tfrac{m}{n} \rceil) > \log_2 n\}.$$

Ackermannova funkce A je velmi rychle rostoucí funkce, která není primitivně rekurzivní. Následující tabulka ukazuje její hodnoty pro malá čísla i a x.

A	0	1	2	3	4	5	...	x
0	0	2	4	6	8	10	...	$2x$
1	0	2	4	8	16	32	...	2^x
2	0	2	4	16	65536	2^{65536}	...	$2^{\cdot^{\cdot^{\cdot^{2}}}}$ $x-$krát
3	0	2	4	65536	$2^{\cdot^{\cdot^{\cdot^{2}}}}$ 65536 − krát			

Odtud je vidět, že když $\log n < A(3,4)$, pak $\alpha(m,n) \leq 3$, což znamená, že pro všechna představitelná čísla m a n je $\alpha(m,n) \leq 3$. Jak uvidíme z následující věty, modifikovaný algoritmus má tedy v reálných situacích prakticky lineární složitost.

Věta 11.3. *Realizace posloupnosti obsahující m operací* **FIND** *a $n-1$ operací* **UNION**, *kde $m > n$, v modifikované datové struktuře vyžaduje čas $O(m\alpha(m,n))$.*

Důkaz. Pro jednoduchost budeme ztotožňovat vrchol stromu s prvkem, který reprezentuje. Abychom dokázali naše tvrzení, budeme nejprve modifikovat naši posloupnost operací $\mathfrak{P}$. Předpokládejme, že argumentem k-té operace **FIND** v posloupnosti $\mathfrak{P}$ je prvek x_k, a nechť y_k je prvek reprezentovaný v kořeni toho stromu, který v okamžiku provádění k-té operace **FIND** obsahuje x_k.

Zavedeme následující modifikaci operace **FIND**. Když prvky $x, y \in U$ jsou reprezentovány ve stromě T tak, že platí, že y leží na cestě z x do kořene stromu, pak definujme operaci **pFIND**(x, y). Tato operace projde (vyšetří) všechny hrany na cestě z vrcholu x do vrcholu y (v pořadí od vrcholu x k vrcholu y) a všechny vrcholy na této cestě předcházející vrcholu y přemění na syny y. Předpokládáme, že tato operace vyžaduje čas rovný počtu vrcholů na této cestě.

Naším cílem bude ukázat, že posloupnost $\mathfrak{P}$ vyžaduje stejný čas a vytvoří stejný strom jako posloupnost $\mathfrak{P}'$, kterou získáme tak, že vezmeme podposloupnost $\mathfrak{P}_U$ posloupnosti $\mathfrak{P}$ tvořenou všemi operacemi **UNION** v posloupnosti $\mathfrak{P}$ (tedy vynecháme všechny operace **FIND**) a vezmeme posloupnost $\mathfrak{P}_F$ tvořenou všemi operacemi **FIND** v posloupnosti $\mathfrak{P}$ (tedy vynecháme všechny operace **UNION**). Vytvoříme posloupnost $\mathfrak{P}'_F$ z posloupnosti $\mathfrak{P}_F$ tak, že k-tou operaci **FIND**(x_k) nahradíme operací **pFIND**(x_k, y_k). Posloupnost $\mathfrak{P}'$ je pak konkatenace posloupnosti $\mathfrak{P}_U$ a posloupnosti $\mathfrak{P}'_F$.

Abychom to dokázali, proveďme posloupnost $\mathfrak{P}$ a označme $\{T_i \mid i \in I_k\}$ soubor stromů reprezentující naši strukturu před provedením k-té operace **FIND**(x_k) a $\{V_i \mid i \in I_k\}$ soubor stromů po provedení operace **FIND**(x_k). Nyní na tyto soubory aplikujme posloupnost $\mathfrak{P}_k$, kterou získáme z posloupnosti $\mathfrak{P}$ vynecháním všech operací **UNION** před k-tou operací **FIND**(x_k) a všech operací **FIND**. Když provedeme $\mathfrak{P}_k$ na soubor $\{T_i \mid i \in I_k\}$, dostaneme strom T_{k-1}, a když provedeme $\mathfrak{P}_k$ na soubor $\{V_i \mid i \in I_k\}$, dostaneme strom T_k. Je lehce vidět, že operaci **pFIND**(x_k, y_k) lze provést na strom T_{k-1} (tj. vrchol y_k ve stromě T_{k-1} leží na cestě z vrcholu x_k do kořene) a tato operace vyžaduje stejný čas jako operace **FIND**(x_k) aplikovaná na soubor $\{T_i \mid i \in I_k\}$. Výsledkem operace **pFIND**(x_k, y_k) pak je strom T_k.

Z definice plyne, že provedením operace $\mathfrak{P}_U$ na iniciální strukturu dostaneme strom T_0. Indukcí z předchozího pozorování plyne, že když T_k je strom po provedení k operací v posloupnosti $\mathfrak{P}'_F$, pak tento strom dostaneme tak, že provedeme posloupnost $\mathfrak{P}$ až do k-té operace **FIND**(x_k) včetně této operace, a pak aplikujeme posloupnost $\mathfrak{P}_k$. Z toho plyne, že obě posloupnosti $\mathfrak{P}$ a $\mathfrak{P}'$ dávají stejný strom a vyžadují stejný čas.

Označme F množinu všech hran, které byly vyšetřovány během realizace posloupnosti $\{\mathbf{pFIND}(x_k, y_k)\}_{k=1}^m$, a u každé hrany $(u, v) \in F$ (předpokládáme otec$(u) = v$) označme $\nu(u, v)$ počet operací, které tuto hranu vyšetřovaly. Pak čas na realizaci posloupnosti $\{\mathbf{pFIND}(x_k, y_k)\}_{k=1}^m$ ve stromě T_0 je úměrný $\sum_{(u,v)\in F} \nu(u, v)$. Naším cílem bude odhadnout tento součet. Všimněme si, že každá hrana, která byla vyšetřována v nějaké operaci **pFIND** a nebyla poslední vyšetřovanou hranou, přestala být hranou stromu, a tedy už žádná následující operace **pFIND** ji nemůže vyšetřovat. Odhad součtu $\sum_{(u,v)\in F} \nu(u, v)$ vyžaduje přeformulování tohoto problému pomocí následujících pojmů. Označme vys(v) výšku vrcholu v ve stromě T_0 (tj. délku nejdelší cesty z vrcholu v do listu v podstromu T_0 určeném vrcholem v). Dále definujme

$$G_{i,j} = \{u \mid A(i, j) \leq \text{vys}(u) < A(i, j + 1)\}.$$

Nejprve odhadneme velikost $G_{i,j}$

Lemma 11.4. $|G_{i,j}| \leq 2n/2^{A(i,j)}$.

Důkaz. Podle předpokladu mají stromy jen n vrcholů. Uvažujme l takové, že $A(i,j) \leq l < A(i,j+1)$. Odhadneme počet vrcholů u takových, že $\mathrm{vys}(u) = l$. Vezměme takový vrchol u a uvažujme situaci, že u je kořen nějakého stromu, ale po provedení následující operace **UNION** už není kořenem žádného stromu. Když vrchol není kořen, tak žádná operace **UNION** nepřidává nic do jeho podstromu, a proto už v tomto okamžiku musel mít výšku l, a tedy v jeho podstromu existoval vrchol v, který měl hloubku l. Protože se operace **UNION** v modifikované struktuře nezměnila, musí platit invariant z důkazu Věty 11.2, a tedy strom, jehož kořenem byl vrchol u, musel mít alespoň 2^l vrcholů. Každé dva různé vrcholy u a u' takové, že $\mathrm{vys}(u) = \mathrm{vys}(u')$, mají disjunktní podstromy. Odtud plyne, že je nejvýše $\frac{n}{2^l}$ vrcholů u takových, že $\mathrm{vys}(u) = l$. Tedy dostáváme

$$|G_{i,j}| \leq \sum_{l=A(i,j)}^{A(i,j+1)-1} \frac{n}{2^l} \leq \frac{2n}{2^{A(i,j)}}. \quad \square$$

Protože $A(i,0) = 0$ a $A(i,j) < A(i,j+1)$ pro každé $i = 0,1,\ldots$ a každé $j = 0,1,\ldots$, dostáváme, že pro každé $i = 0,1,\ldots$ jsou množiny $G_{i,0}, G_{i,1}, \ldots$ navzájem disjunktní a jejich sjednocením je celé U.

Mějme parametr z, který budeme specifikovat později, a definujme

$$N_k = \{(x,y) \in F \mid k = \min\{i \mid \exists j,\, x,y \in G_{i,j}\}\} \qquad \text{pro } 0 \leq k \leq z$$

a $N_{z+1} = F \setminus \bigcup_{k=0}^{z} N_k$. Všimněme si, že pro $\{x,y\} \in N_k$, $k \leq z$ existuje j takové, že $\mathrm{vys}(x) < A(k-1,j) \leq \mathrm{vys}(y)$. Dále pro $k = 0,1,\ldots,m$ označme

$$M_k = \{((x,y),i) \mid (x,y) \in N_k, \text{operace } \mathbf{pFIND}(x_i,y_i) \text{ procházela hranu } (x,y)\}.$$

Protože operace **pFIND** prochází každou hranu nejvýše jednou, platí $\sum_{(x,y)\in F} \nu(x,y) = \sum_{k=0}^{z+1} |M_k|$. Nechť L_k je množina všech prvků $((x,y),i) \in M_k$ takových, že (x,y) je poslední hrana v N_k, kterou prochází operace $\mathbf{pFIND}(x_i,y_i)$. Nyní odhadneme velikosti množin L_k a $M_k \setminus L_k$. Označme $a(z,n) = \min\{i \mid A(z,i) > \log n\}$. Pak

Lemma 11.5. *Platí*

(1) $|L_k| \leq m$ *pro každé* k *takové, že* $0 \leq k \leq z+1$;
(2) $|M_0 \setminus L_0| \leq n$;
(3) $|M_k \setminus L_k| \leq \frac{5n}{8}$ *pro každé* k *takové, že* $1 \leq k \leq z$;
(4) $|M_{z+1} \setminus L_{z+1}| \leq na(z,n)$.

Důkaz. (1) plyne z faktu, že pro každé $i = 1,2,\ldots,m$ platí, že když $((x,y),i), ((x',y'),i) \in L_k$, pak $(x,y) = (x',y')$.

Dokážeme (2). Mějme prvek $((u,v),i) \in M_0 \setminus L_0$. Pak existuje j takové, že $A(0,j) = 2j \leq \mathrm{vys}(u) < \mathrm{vys}(v) < 2j+2 = A(0,j+1)$. Z definice L_0 plyne existence $((s,t),i) \in L_0$. Pak platí $\mathrm{vys}(v) \leq \mathrm{vys}(s) < \mathrm{vys}(t) \leq \mathrm{vys}(y_i)$. Odtud dostáváme, že $\mathrm{vys}(y_i) > A(0,j+1)$, a proto už pro žádné $i' > i$ neexistuje prvek $((u,w),i') \in M_0$. Tedy pro každý vrchol u existuje nejvýše jeden prvek $((u,v'),i')$ v $M_0 \setminus L_0$ a (2) je dokázáno.

Nyní začneme dokazovat (3) a (4). Mějme $k \in \{1, 2, \ldots, z+1\}$ a uvažujme vrchol u takový, že $A(k,j) \leq \mathrm{vys}(u) < A(k,j+1)$. Předpokládejme, že

$$((u,v_1),i_1),((u,v_2),i_2),\ldots,((u,v_q),i_q)$$

jsou všechny prvky v $M_k \setminus L_k$, jejichž první komponenta je u a platí $\mathrm{vys}(v_1) \leq \mathrm{vys}(v_2) \leq \cdots \leq \mathrm{vys}(v_q)$ (nutně platí $\mathrm{vys}(u) < \mathrm{vys}(v_1)$). Zvolme $l \in \{1,2,\ldots,q\}$. Z definice L_k plyne existence $((s_l,t_l),i_l) \in L_k$, a proto platí $\mathrm{vys}(v_l) \leq \mathrm{vys}(s_l) < \mathrm{vys}(t_l) \leq \mathrm{vys}(y_{i_l})$. Z popisu operace pak plyne, že $\mathrm{vys}(y_{i_l}) \leq \mathrm{vys}(v_{l+1})$. Protože $(s_l,t_l) \notin N_{k-1}$, dostáváme existenci j_l takového, že

$$\mathrm{vys}(u) < \mathrm{vys}(v_1) \leq \mathrm{vys}(v_l) \leq \mathrm{vys}(s_l) < A(k-1,j_l) \leq \mathrm{vys}(t_l) \leq \mathrm{vys}(y_{i_l}).$$

Odtud $\mathrm{vys}(v_l) < A(k-1,j_l) \leq \mathrm{vys}(v_{l+1})$ a tedy $j_l < j_{l+1}$. Odtud plyne $j_1 < j_1+q-1 \leq j_{q-1}$. Dále $(u,v_1) \notin N_{k-1}$ a tedy existuje j' takové, že

$$\mathrm{vys}(u) < A(k-1,j') \leq \mathrm{vys}(v_1) < A(k-1,j_1),$$

a proto $j_1 \geq 1$.

V tomto okamžiku rozdělíme důkazy (3) a (4). Nejprve dokončíme důkaz (3). Protože $(u,v_q) \in N_k$ a $u \in G_{k,j}$, dostáváme, že $v_q \in G_{k,j}$, a odtud plyne, že $\mathrm{vys}(y_q) < A(k,j+1) = A(k-1,A(k,j))$. Proto $A(k-1,j_1+q-1) < A(k-1,A(k,j))$, a tedy $j_1+q-1 < A(k,j)$ a odtud $q \leq A(k,j)$ (protože $j_1 \geq 1$). Z $A(0,0) = A(l,0)$ a $A(0,1) = A(l,1)$ pro všechna l plyne $k \geq 1$ a $j \geq 2$. Pak z každého vrcholu $u \in G_{k,j}$ vychází nejvýše $A(k,j)$ hran (u,v) v N_k takových, že $((u,v),i) \in M_k \setminus L_k$ pro nějaké $i \in \{1,2,\ldots,m\}$. Protože pro každou hranu $(u,v) \in N_k$ existuje nejvýše jedno $i \in \{1,2,\ldots,m\}$ takové, že $((u,v),i) \in M_k \setminus L_k$, tak z Lemmatu 11.4 plyne

$$|M_k \setminus L_k| \leq \sum_{j=2}^{\infty} |G_{k,j}|A(k,j) \leq \sum_{j=2}^{\infty} \frac{2nA(k,j)}{2^{A(k,j)}} \leq \sum_{j=2}^{\infty} \frac{2n2^j}{2^{2^j}} \leq \frac{5n}{8}.$$

Vysvětlení: Druhá nerovnost plyne z Lemmatu 11.4 a třetí nerovnost plyne z faktu, že za uvedených podmínek platí $A(i,j) \geq 2^j$ a funkce $\frac{x}{2^x}$ je klesající v intervalu $(2,+\infty)$ (použijeme pro $x = A(k,j)$). Abychom dostali čtvrtou nerovnost, uvědomme si, že $2j+2-2^j \leq 0$ pro $j \geq 3$. Proto pro $j \geq 3$ platí

$$1 \geq 2^{2j+2-2^j} = \frac{2^{2j+2}}{2^{2^j}} = \frac{2^{j+1}2^{j+1}}{2^{2^j}}$$

a odtud $\frac{2^{j+1}}{2^{2^j}} \leq \frac{1}{2^{j+1}}$. Nyní použijeme $\sum_{j=4}^{\infty} \frac{1}{2^j} = \frac{1}{8}$ a $\frac{2(2^2)}{2^{2^2}} = \frac{1}{2}$, abychom dostali, že $\sum_{j=2}^{\infty} \frac{2(2^j)}{2^{2^j}} = \sum_{j=2}^{\infty} \frac{2^{j+1}}{2^{2^j}} \leq \frac{5}{8}$. Tím je důkaz (3) kompletní.

Nyní se vrátíme k důkazu (4). V důkazu Lemmatu 11.4 jsme ukázali, že $\mathrm{vys}(v_q) \leq \log n$. Proto $A(z,j_1+q-1) \leq \log n$. Víme, že $k-1 = z$ (předpokládáme $k = z+1$), a tak z definice $a(z,n)$ plyne $j_1+q-1 < a(z,n)$ (protože $j_1 \geq 1$) a odtud $q \leq a(z,n)$. Tedy z každého vrcholu v $u \in G_{z+1,j}$ vychází nejvýše $a(z,n)$ hran $(u,v) \in N_{z+1}$. Z faktu, že pro každou hranu

$(u,v) \in N_{z+1}$ existuje nejvýše jedno $i \in \{1,2,\ldots,m\}$ takové, že $((u,v),i) \in M_{z+1} \setminus L_{z+1}$, dostáváme $|M_{z+1} \setminus L_{z+1}| \leq na(zn)$, a tím je (4) dokázáno. □

Z těchto výsledků okamžitě plyne

$$\sum_{(u,v)\in F} \nu(u,v) = \sum_{i=0}^{z+1} |L_i| + \sum_{i=0}^{z+1} |M_i \setminus L_i| \leq m(z+2) + n + \frac{5n}{8}z + na(z,n).$$

Když zvolíme $z = \min\{i \geq 1 \mid A(i, 4\lceil \frac{m}{n} \rceil) > \log n\} = \alpha(m,n)$ a když si uvědomíme, že

$$a(z,n) = a(\alpha(m,n),n) = \min\{x \mid A(\alpha(m,n),x) > \log n\} \leq 4\lceil \frac{m}{n} \rceil \leq \frac{8m}{n},$$

dostaneme, že

$$\sum_{(u,v)\in F} \nu(u,v) \leq m(z+2)+n+\frac{5n}{8}z+na(z,n) \leq (m+\frac{5n}{8})\alpha(m,n)+n+10m = O(m\alpha(m,n)),$$

protože $m \geq n$, a Věta 11.3 je dokázána. □

Výsledky Vět 11.1 a 11.2 jsou důsledkem idejí, které prezentovali Aho, Hopcroft a Ullman v roce 1973. Větu 11.3 dokázal v roce 1975 Tarjan. Tarjan také ukázal, že téměř každý algoritmus řešící **UNION-FIND** problém potřebuje alespoň $\Omega(m\alpha(m,n))$ času. Tedy za jistých velmi rozumných předpokladů je prezentovaný algoritmus až na multiplikativní konstantu optimální (neboli problém **UNION-FIND** sice nelze řešit v lineárním čase, ale pro prakticky použitelné vstupy je algoritmus lineární).

12. Volba ze dvou

V první přednášce zimního semestru jsme hledali horní odhad na očekávanou délku maximálního řetězce pro hašování se separovanými řetězci, když rozdělení vstupních dat bylo rovnoměrné. Tento problém lze přeformulovat následujícím způsobem:

> Máme m přihrádek a n míčků, kde $n \leq m$, a všechny míčky chceme umístit do těchto přihrádek (není omezen počet míčků v jedné přihrádce). Míčky umisťujeme postupně a když chceme míček umístit do přihrádek tak náhodně, s rovnoměrným rozdělením, volíme přihrádku, kam míček vložíme. Náš úkol byl nalézt odhad na očekávaný maximální počet míčků v jedné přihrádce.

V zimním semestru jsme si ukázali, že očekávaný maximální počet míčků v jedné přihrádce je $O(\frac{\log n}{\log \log n})$. V této přednášce budeme studovat modifikaci tohoto problému a jeho důsledky pro datové struktury. Při vkládání míčku do přihrádky nevolíme jednu přihrádku, ale zvolíme náhodně s rovnoměrným rozdělením a nezávisle d přihrádek, kde $d \geq 2$ je fixované číslo. Z těchto zvolených přihrádek vezme ty, které obsahují nejméně míčků a do první z těchto přihrádek míček vložíme. Náš úkol je stejný jako v původním problému, tj. nalézt očekávaný maximalní počet míčků v jedné přihrádce.

Nejprve si uvedeme dvě pomocné tvrzení. První je specifická verze Chernoffovy nerovnosti.

Lemma 12.1. *Když Z je binomická náhodná proměnná s parametry n a p, pak*

$$\mathrm{Prob}(Z \geq 2np) \leq e^{-\frac{np}{3}}.$$

□

Než uvedeme druhé tvrzení připomeneme si několik pojmů a faktů z teorie pravděpodobnosti. Bernoulliho proces je posloupnost nezávislých náhodných proměnných $X_1, X_2, \ldots, X_n$ jejichž obor hodnot je $\{0, 1\}$ a mají stejnou distribucí. To znamená, že Bernoulliho proces je určen počtem proměnných n a pravděpodobností $p = \mathrm{Prob}(X_i = 1)$ (protože proměnné mají stejnou distribuci, tak p není závislé na i). Součet těchto proměnných dává náhodnou proměnnou s binomiálním rozdělením určeným n a p. Proměnnou nazveme binární když obor jejich hodnot je $\{0, 1\}$

Lemma 12.2. *Když $X_1, X_2, \ldots, X_n$ je posloupnost náhodných proměnných a $Y_1, Y_2, \ldots, Y_n$ je posloupnost náhodných binárních proměnných takových, že $Y_i = Y_i(X_1, X_2, \ldots, X_i)$ a*

$$\mathrm{Prob}(Y_i = 1 \mid X_1, X_2, \ldots, X_{i-1}) \leq p$$

pro každé i, pak

$$\mathrm{Prob}(\sum_{i=1}^{n} Y_i > k) \leq \mathrm{Prob}(Z > k),$$

kde Z je binomická náhodná proměnná s parametry n a p.

Důkaz. Vezměme Bernoulliho proces $I_1, I_2, \ldots, I_n$ s n proměnnými a s pravděpodobností p. Pak pro každé i platí

$$\mathrm{Prob}(Y_i = 1) \leq \mathrm{Prob}(I_i = 1)$$

a jednoduchou indukcí dostaneme

$$\mathrm{Prob}(\sum_{i=1}^{n} Y_i > k) \leq \mathrm{Prob}(\sum_{i=1}^{n} I_i > k).$$

Teď si stačí připomenout, že součet n proměnných v Bernoulliho procesu s pravděpodobností p dává náhodnou proměnnou s binomickým rozdělením určeným parametry n a p a dostaneme požadované tvrzení. □

V následující analýze budeme předpokládat, že $n = m$ a přihrádky tvoří zásobník (bez operace pop). To znamená, že i-tý vložený míček do k-té přihrádky bude v ní na i-té pozici. Předpokládejme, že máme posloupnost $\beta_1, \beta_2, \ldots, \beta_n$ čísel takovou, že β_i je s "velkou pravděpodobnostní" horní odhad na počet příhrádek, které po vložení všech míčků do přihrádek, obsahují alespoň i míčků. Dále označme $\nu_i(t)$ počet přihrádek po vložení t míčků obsahují alespoň i míčků, $\mu_i(t)$ počet všech míčků, které jsou po vložení t míčků ve své přihrádce na j-té pozici pro $j \geq i$ a $h(t)$ označme pozici t-tého míčku (tj. míčku vkládaného, jako t-tý) v jeho přihrádce. Místo $\nu_i(n)$ a $\mu_i(n)$ budeme psát jen ν_i a μ_i. Všimněme si, že $\nu_i(t) \leq \mu_i(t)$ a s "velkou pravděpodobností" bude platit $\nu_i \leq \beta_i$.

Nejprve uvažujme, že pro dané i je β_i skutečně horní odhad na ν_i. Pak pravděpodobnost, že libovolná fixovaná přihrádka má alespoň i míčků, je nejvýše $\frac{\beta_i}{n}$. Tedy pravděpodobnost, že

$h(t) > i$ je nejvýše $\left(\frac{\beta_i}{n}\right)^d$. Proto očekávaná hodnota μ_{i+1} je nejvýše $n\left(\frac{\beta_i}{n}\right)^d$ a z Markovovy nerovnosti plyne, že $\beta_{i+1} \leq 2n\left(\frac{\beta_i}{n}\right)^d$ s pravděpodobností větší než $\frac{1}{2}$. Tato úvaha nás vede k tomu, že položíme $\beta_{i+1} = 2n\left(\frac{\beta_i}{n}\right)^d$ pro $4 \leq i \leq i^*$, kde i^* budeme specifikovat později. Dále, když položíme $\beta_4 = \frac{n}{4}$, pak s pravděpodobností 1 platí, že $\nu_4 \leq \beta_4$. Proto budeme můžeme předpokládat, že s velkou pravděpodobností platí $\nu_i = \nu_i(n) \leq \beta_i$ pro všechna $i \leq i^*$. Označme $\mathcal{E}_i$ událost, že $\nu_i(n) \leq \beta_i$. Tedy $\mathcal{E}_4$ platí s pravděpodobností 1 a ukážeme, že s velkou pravděpodobností z platnosti $\mathcal{E}_i$ plyne platnost $\mathcal{E}_{i+1}$ pro $4 \leq i < i^*$.

Zafixujme i z daného definičního intervalu a definujme binární proměnnou Y_t tak, že $Y_t = 1$ právě, když $h(t) \geq i+1$ a $\nu_i(t-1) \leq \beta_i$. Tedy $Y_t = 1$, když se t-tý míček vkládá do přihrádky na pozici aspoň $i+1$ a po vložení $t-1$ míčků má nejvýše β_i přihrádek alespoň i míčků. Tedy

$$\mathrm{Prob}(Y_t = 1 \mid (\nu_i(t-1) \leq \beta_i)) \leq \left(\frac{\beta_i}{n}\right)^d.$$

Tento odhad podmíněné pravděpodobností využívá faktu, že $Y_t = 0$, když více než β_i přihrádek má alespoň i míčků. Dále, protože míčky jsou vkládány nezávisle, tak proměnné Y_t budou nezávislé a tedy jsou splněné předpoklady Lemmatu 12.2.

Položme $p_i = \left(\frac{\beta_i}{n}\right)^d$, a nechť Z_i je binomická náhodná proměnná s parametry n a p_i. Pak podle Lemmatu 12.2 dostáváme

$$\mathrm{Prob}\left(\sum_{t=1}^{n} Y_t > k\right) \leq \mathrm{Prob}(Z_i > k).$$

Vzhledem k naší definici Y_t toto platí nezávisle na události $\mathcal{E}_i$. Protože $\nu_{i+1} \leq \mu_{i+1}$ a při události $\mathcal{E}_i$ platí $\sum_{t=1}^{n} Y_t = \mu_{i+1}$, tak dostáváme

$$\begin{aligned}\mathrm{Prob}(\nu_{i+1} > k \mid \mathcal{E}_i) \leq & \mathrm{Prob}(\mu_{i+1} > k \mid \mathcal{E}_i) = \\ & \mathrm{Prob}\left(\sum_{t=1}^{n} Y_t > k \mid \mathcal{E}_i\right) \leq \\ & \frac{\mathrm{Prob}\left(\sum_{t=1}^{n} Y_t > k\right)}{\mathrm{Prob}(\mathcal{E}_i)} \leq \\ & \frac{\mathrm{Prob}(Z_i > k)}{\mathrm{Prob}(\mathcal{E}_i)}.\end{aligned}$$

Nyní použijeme Lemma 12.1 pro $k = \beta_{i+1} = 2np_i$ a dostaneme

$$\mathrm{Prob}(\nu_{i+1} > k \mid \mathcal{E}_i) \leq \frac{\mathrm{Prob}(Z_i > 2np_i)}{\mathrm{Prob}(\mathcal{E}_i)} \leq \frac{1}{e^{\frac{p_i n}{3}} \mathrm{Prob}(\mathcal{E}_i)}.$$

Protože $\nu_{i+1} > k$ znamená, že nenastala událost $\mathcal{E}_{i+1}$, tak pokud platí $p_i n \geq 6 \ln n$, tak dostáváme, že

$$\mathrm{Prob}(\neg\mathcal{E}_{i+1} \mid \mathcal{E}_i) \leq \frac{1}{n^2 \mathrm{Prob}(\mathcal{E}_i)}.$$

Nyní odstraníme podmíněnou pravděpodobnost standardním postupem

$$\begin{aligned}\mathrm{Prob}(\neg\mathcal{E}_{i+1}) = &\mathrm{Prob}(\neg\mathcal{E}_{i+1} \mid \mathcal{E}_i)\,\mathrm{Prob}(\mathcal{E}_i) + \\ &\mathrm{Prob}(\neg\mathcal{E}_{i+1} \mid \neg\mathcal{E}_i)\,\mathrm{Prob}(\neg\mathcal{E}_i) \leq \\ &\mathrm{Prob}(\neg\mathcal{E}_{i+1} \mid \mathcal{E}_i)\,\mathrm{Prob}(\mathcal{E}_i) + \mathrm{Prob}(\neg\mathcal{E}_i).\end{aligned}$$

Odtud plyne, že

$$\mathrm{Prob}(\neg\mathcal{E}_{i+1}) \leq \mathrm{Prob}(\neg\mathcal{E}_i) + \frac{1}{n^2},$$

když $p_i n \geq 6 \ln n$. Tedy když $p_i n \geq 6 \ln n$ a s velkou pravděpodobností platí $\mathcal{E}_i$, pak s velkou pravděpodobností platí také $\mathcal{E}_{i+1}$.

Abychom dokončili důkaz, ukážeme, že když i^* je nejmenší i takové, že $p_i n < 6 \ln n$, pak i^* je $\frac{\ln \ln n}{\ln d} + O(1)$. Nejprve si všimněme, že pro každé $i < i^*$ platí $p_i n \geq 6 \ln n$ a tedy z předchozího odhadu dostaneme indukcí podle i, že

$$\mathrm{Prob}(\neg\mathcal{E}_{i^*}) \leq \frac{i^* - 4}{n^2},$$

protože $\mathrm{Prob}(\neg\mathcal{E}_4) = 0$. Následující odhad β_{i+4}, který dokážeme indukcí podle i, nám dá odhad na i^*

$$\beta_{i+4} = \frac{n}{2^{2d^i - \sum_{j=0}^{i-1} d^j}}.$$

Tvrzení zřejmě platí pro $i = 0$. Dále platí

$$\begin{aligned}\beta_{(i+1)+4} = &\frac{2\beta_{i+4}^d}{n^{d-1}} = \frac{2\left(\frac{n}{2^{2d^i - \sum_{j=0}^{i-1} d^j}}\right)^d}{n^{d-1}} = \\ &\frac{2n}{2^{2d^{i+1} - \sum_{j=1}^{i} d^i}} = \frac{n}{2^{2d^{i+1} - \sum_{j=0}^{i} d^i}},\end{aligned}$$

kde první rovnost je definice $\beta_{(i+1)+4}$, druhá rovnost využívá indukčního předpokladu.

Z tohoto výrazu dostáváme, že $\beta_{i+4} \leq \frac{n}{2^{d^i}}$, protože $d^i \geq \frac{d^i - 1}{d-1} = \sum_{j=0}^{i-1} d^j$. Protože $p_i = \frac{\beta_i^d}{n^d}$ tak dostáváme, že

$$p_{i+4} = \frac{\beta_{i+4}^d}{n^d} \leq \frac{\frac{n^d}{\left(2^{d^i}\right)^d}}{n^d} = \frac{1}{2^{d^{i+1}}}.$$

Když $\frac{n}{2^{d^{i+1}}} < 6 \ln n$, pak $p_{i+4} n < 6 \ln n$ a tedy $i^* < i+4$. Vztah $\frac{n}{2^{d^{i+1}}} < 6 \ln n$ je ekvivalentní se vztahem $2^{d^{i+1}} > \frac{n}{6 \ln n}$ a logaritmovaním postupně dostáváme, že

$$\begin{aligned}d^{i+1} \ln 2 &> \ln n - \ln 6 - \ln \ln n \\ (i+1) \ln d + \ln \ln 2 &> \ln(\ln n - \ln 6 - \ln \ln n).\end{aligned}$$

Z toho plyne, že pro $i+1 > \frac{\ln\ln n}{\ln d}$ platí $p_{i+4}n < 6\ln n$, a proto $i^* = \frac{\ln\ln n}{\ln d} + O(1)$.
Nyní vyšetříme případ $p_i n < 6\ln n$. Podle Lemmatu 12.1 dostaneme, že

$$\begin{aligned}\mathrm{Prob}(\nu_{i^*+1} > 12\ln n \mid \mathcal{E}_{i^*}) \le \mathrm{Prob}(\mu_{i^*+1} > 12\ln n \mid \mathcal{E}_{i^*}) \le \\ \frac{\mathrm{Prob}(Z \le 12\ln n)}{\mathrm{Prob}(\mathcal{E}_{i^*})} \le \\ \frac{1}{n^2\,\mathrm{Prob}(\mathcal{E}_{i^*})},\end{aligned}$$

kde Z je binomická náhodná proměnná s parametry n a $\frac{6\ln n}{n}$.

Nyní odstraníme stejně jako v předchozím případě podmíněnou pravděpodobnost, když použijeme předchozí odhad a fakt, že posloupnost $\{p_i\}_{i=1}^{\infty}$ je klesající

$$\mathrm{Prob}(\nu_{i^*+1} > 12\ln n) \le \mathrm{Prob}(\neg\mathcal{E}_{i^*}) + \frac{1}{n^2} \le \frac{i^*-3}{n^2}.$$

Zřejmě platí

$$\mathrm{Prob}(\nu_{i^*+3} \ge 1) \le \mathrm{Prob}(\mu_{i^*+3} \ge 1) \le \mathrm{Prob}(\mu_{i^*+2} \ge 2).$$

Dále platí

$$\begin{aligned}\mathrm{Prob}(\mu_{i^*+2} \ge 2 \mid \nu_{i^*+1} \le 12\ln n) \le \frac{\mathrm{Prob}(Z' \ge 2)}{\mathrm{Prob}(\nu_{i^*+1} \le 12\ln n)} \le \\ \frac{\binom{n}{2}(\frac{12\ln n}{n})^{2d}}{\mathrm{Prob}(\nu_{i^*+1} \le 12\ln n)},\end{aligned}$$

kde Z' je binomická proměnná s parametry n a $\left(\frac{12\ln n}{n}\right)^d$ a poslední nerovnost získáme hrubou silou, protože je $\binom{n}{2}$ dvojic míčků a pro každou dvojici, pravděpodobnost, že ve své přihrádce mají pozici aspoň i^*+2 je $\left(\frac{12\ln n}{n}\right)^{2d}$.

Spojením těchto odhadů dostaneme

$$\begin{aligned}\mathrm{Prob}(\nu_{i^*+3} \ge 1) \le \mathrm{Prob}(\mu_{i^*+2} \ge 2) \le \\ \mathrm{Prob}(\mu_{i^*+2} \ge 2 \mid \nu_{i^*+1} \le 12\ln n)\,\mathrm{Prob}(\nu_{i^*+1} \le 12\ln n) + \\ \mathrm{Prob}(\nu_{i^*+1} > 12\ln n) \le \\ \frac{(12\ln n)^{2d}}{n^{2d-2}} + \frac{i^*-3}{n^2},\end{aligned}$$

kde jsme použili, že $\binom{n}{2} \le n^2$. Odtud plyne, že $\mathrm{Prob}(\nu_{i^*+3} \ge 1)$ je $o(\frac{1}{n})$ pro $d \ge 2$ a z toho plyne, že maximální počet míčků v jedné přihrádce je větší než $i^*+3 = \frac{\ln\ln n}{\ln d} + O(1)$ má pravděpodobnost $o(\frac{1}{n})$.

Věta 12.3. *Předpokládejme, že máme n přihrádek a n míčků a že $d \ge 2$ je přirozené číslo. Vkládáme postupně míčky do přihrádek tak, že zvolíme náhodně s rovnoměrným rozdělením a nezávisle d přihrádek a míček vložíme do nejméně zaplněné přihrádky. Po vložení všech*

míčků je maximální počet míčků v jedné přihrádce nejvýše $\frac{\ln\ln n}{\ln d} + O(1)$ *s pravděpodobností aspoň* $1 - o(\frac{1}{n})$. □

Podobnou technikou lze ukázat, že tento výsledek nelze vylepšit. Uvedeme schema důkazu. Definujme γ_i tak, že $\gamma_0 = n$ a $\gamma_{i+1} = \frac{n}{2^{i+3}} \left(\frac{\gamma_i}{n}\right)^d$ a nechť $\mathcal{F}_i$ je událost, že $\nu_i(n(1 - \left(\frac{1}{2}\right)^i)) \geq \gamma_i$. Položme $p_i = \frac{1}{2}\left(\frac{\gamma_i}{n}\right)^d$ a nechť i^* je dolní mez pro čísla i, pro které platí $\frac{p_i n}{2^{i+1}} \geq 17 \ln n$. Pak podobným početním postupem jako pro horní odhad dostaneme, že

$$\mathrm{Prob}(\mathcal{F}_{i^*}) \geq (1 - \frac{1}{n^2})^{i^*} = 1 - o(\frac{1}{n})).$$

Lze ukázat podobným postupem jako pro β_i, že

$$\gamma_i = \frac{n}{2^{\sum_{k=0}^{i-1}(i+2-k)d^k}},$$

a tedy $\gamma_i \geq \frac{n}{2^{10 d^{i-1}}}$. Z toho dostaneme, že pro dostatečně velká n a pro i takové, že $d^{i-1} \leq \frac{1}{20}\ln n$ platí $\gamma_i \geq 17 \ln n$ a z toho dostaneme, že $i^* = \frac{\ln\ln n}{\ln d} - O(1)$ a tím je dokázáno, že je malá pravděpodobnost, aby platil menší odhad na maximální počet míčků v jedné přihrádce.

Když bychom chtěli použít tento výsledek přímo pro hašování potřebovali bychom d funkcí, které prvkům univerza přiřazují d-tice čísel z množiny $\{1, 2, \ldots, m\}$ tak, že když vybereme n prvků pro $n \leq m$, pak získáme d-tice s rovnoměrným rozdělením a prvky v d-tice jsou navzájem nezávislé. Přitom funkce musíme umět rychlé vyčíslit. Toto je problém zajistit. Univerzální hašování nám však dává možnost jak výsledek použít.

Musíme modifikovat pojem univerzálního soubor funkcí. Řekneme, že soubor funkcí $\mathcal{H} = \{h_i \mid i \in I\}$ z univerza U do čísel $\{0, 1, \ldots, m-1\}$ je silně k-univerzální soubor funkcí pro přirozené číslo $k \geq 1$, když pro každou prostou posloupnost prvků $\{x_i\}_{i=1}^k$ z univerza a každou posloupnost prvků $\{y_i\}_{i=1}^k$ z množiny $\{0, 1, \ldots, m-1\}$ platí

$$|\{i \in I \mid \forall j = 1, 2, \ldots, k,\ h_i(x_j) = y_j\}| = \frac{|I|}{m^k}.$$

Toto je ekvivaletní s tvrzením, že pro každou prostou posloupnost $\{x_i\}_{i=1}^k$ prvků z univerza a pro každou posloupnost $\{y_i\}_{i=1}^k$ prvků z množiny $\{0, 1, \ldots, m-1\}$ platí

$$\mathrm{Prob}_{f \in \mathcal{H}}(f(x_1) = y_1, f(x_2) = y_2, \ldots, f(x_k) = y_k) = \frac{1}{m^k}.$$

Použijeme silně 1-univerzální systém $\mathcal{H}$. Nejprve si však všimněme, že není úplně jasná souvislost mezi silně 1-univerzálními systémy a 1-univerzálními systémy. Dále, když máme systém funkcí $\mathcal{H} = \{f_i \mid i \in I\}$ takový, že pro každé $x \in U$ a $y \in \{0, 1, \ldots, m-1\}$ platí $|\{i \in I \mid f_i(x) = y\}| \leq \frac{|I|}{m}$, pak $\mathcal{H}$ je silně 1-univerzální. Skutečně, když existuje $\hat{x} \in U$ a $\hat{y} \in \{0, 1, \ldots, m-1\}$ takové, že $|\{i \in I \mid f_i(\hat{x}) = \hat{y}\}| < \frac{|I|}{m}$, pak

$$|I||U| = \sum_{x,\, y} |\{i \in I \mid f_i(x) = y\}| < |U| m \frac{|I|}{m} = |U||I|,$$

kde $x \in U$ a $y \in \{0, 1, \ldots, m-1\}$ – to je spor.

Nyní vybereme náhodně s rovnoměrným rozdělením a na sobě nezávislých d funkcí $f_1, f_2, \ldots, f_d$ a použijeme následující operace.

Operace **INSERT**(x) nalezne nejmenší i takové, že řetězec $f_i(x)$ je nejkratší mezi řetězci

$$f_1(x), f_2(x), \ldots, f_d(x)$$

a do tohoto řetězce přidáme x.

Operace **MEMBER**(x) prohledává řetězce

$$f_1(x), f_2(x), \ldots, f_d(x)$$

dokud nenalezne x. Pokud neuspěje, pak x není v reprezentované množině.

Pokud vstupní množina je náhodná s rovnoměrným rozdělením, pak podle předchozího výsledku je délka všech řetězců menší než $\frac{\ln \ln n}{\ln d} + O(1)$ s pravděpodobností větší než $1 - o(\frac{1}{n})$, kde n je velikost reprezentované množiny.

Je známo, že předpoklad, že vstup má rovnoměrné rozdělení lze nahradit předpokladem, že budeme vybírat ze silně $\lceil \log n \rceil$-univerzálního souboru. To by však např. znamenalo, že bychom museli dopředu, před vložením prvního prvku museli znát velikost vstupu.

Navíc silně d-univerzální systémy lze zkonstruovat zkonstruovat pomocí grafových expanderů, ale konstrukce je dost komplikovaná. Druhá možnost je jako systém vzít polynomy stupně $|U| - 1$ pro $|U| = m$, ale to není vhodné pro praktické použití. V dalším textu navrhneme použitelnou konstrukci silně 1-univerzálního systému, ale dřív ještě budeme diskutovat o využitelnosti uvedeného výsledku.

Zde se pravděpodobnost bere jak přes všechny vstupy (aby byl splněn požadavek, že vstupy jsou náhodné), tak přes výběr funkcí (aby byl splněn požadavek, že d adres přiřazených vstupu x je navzájem nezávislých). Zde bohužel výběr funkcí není schopen nahradit náhodnost vstupu, jak je vidět z následujícího příkladu. Vezměme systém $\mathcal{H} = \{f_{a,b}(x) = ((ax + b) \bmod N) \bmod m \mid a, b \in \{0, 1, \ldots, N-1\}$, kde universum je množina čísel $\{0, 1, \ldots, n-1\}$ a m je velikost tabulky. Když vezmeme posloupnost vstupu $a_1 = x, a_2 = 2x, a_3 = 3x, a_4 = 4x$, pak ať volíme jakoukoliv funkci $f \in \mathcal{H}$, tak bude vždy platit $f(a_3) = 2f(a_2) - f(a_1)$, $f(a_4) = f(a_3) + f(a_2) - f(a_1)$. Tedy nebudou splněny předpoklady na rovnoměrné rozdělení adres přiřazených k různým vstupům.

Nechť p je prvočíslo, $k > 1$ je přirozené číslo. Popíšeme konstrukci silně 1-univerzálního systému pro tabulku velikosti p a univerzu $U = \{1, 2, \ldots, p^k - 1\}$. Pak existuje bijekce ϕ mezi prvky $u \in U$ a k-ticema čísel $(u_0, u_1, \ldots, u_{k-1})$ z množiny $\{0, 1, \ldots, p-1\}$ různých od $(0, 0, \ldots, 0)$ taková, že $u = \sum_{i=0}^{k-1} u_i p^i$ pro $\phi(u) = (u_0, u_1, \ldots, u_{k-1})$. Pro $j \in \{0, 1, \ldots, k-1\}$ definujme, že $\phi(u) = (\phi_0(u), \phi_1(u), \ldots, \phi_{k-1}(u))$. Pro $a_0, a_1, \ldots, a_{k-1} \in \{0, 1, \ldots, p-1\}$ definujme zobrazení $h_{a_0 a_1, \ldots, a_{k-1}}$ z U do množiny $\{0, 1, \ldots, p-1\}$ takto

$$h_{a_0, a_1, \ldots, a_{k-1}}(u) = \left(\sum_{i=0}^{k-1} a_i \phi_i(u) \right) \bmod p.$$

Položme

$$\mathcal{H} = \{h_{a_0, a_1, \ldots, a_{k-1}} \mid \forall i = 0, 1, \ldots, k-1, \\ a_i \in \{0, 1, \ldots, p-1\} \in \{0, 1, \ldots, p-1\}\}.$$

Pak platí

Věta 12.4. *$\mathcal{H}$ je silně 1-univerzální systém.*

Důkaz. Z definice $\mathcal{H}$ plyne, že velikost množiny indexů je p^k. Vezměme libovolné $u \in U$ a $k \in \{0, 1, \ldots, p-1\}$. Bez újmy na obecnosti můžeme předpokládat, že $\phi_{k-1}(u) \neq 0$.

Zvolme

$$a_0, a_1, \ldots a_{k-2} \in \{0, 1, \ldots, p-1\},$$

pak rovnice

$$\sum_{i=0}^{k-2} \phi_i(u) a_i + x\phi_{k-1}(u) \equiv k \bmod p$$

má právě jedno řešení, protože celá čísla modulo prvočíslo p tvoří těleso. Tedy pro u existuje právě p^{k-1} funkcí z $f \in \mathcal{H}$ (funkce z různými indexy považujeme za různé) takových, že $f(u) = k$. Protože $\phi(u) \neq (0, 0, \ldots, 0)$, existuje i, že $\phi_i(u) \neq 0$ a tedy stačí zaměnit i a $k-1$, takže $\mathcal{H}$ je silně 1-univerzální, protože $p = \frac{p^k}{p}$. □

Tedy předchozí výsledek lze použít pro libovolné d, když tabulka bude mít velikost p pro nějaké prvočíslo p a univerzum bude mít velikost $p^k - 1$, kde $k > 0$ je nějaké přirozené číslo.

Kukaččí hašování.

Kukaččí hašování je jiné využití předchozí ideje. Autoři položili důraz na rychlou realizaci operace **MEMBER** i v nejhorším případě a uvolnili požadavky na operaci **INSERT**. Je to varianta k perfektnímu hašování. Při použití perfektního hašování je problém s operací **INSERT**. To řešila dynamická verze perfektního hašování, kde operace **INSERT** a **DELETE** vyžadovaly konstantní očekávaný amortizovaný čas. Použití ideje z předchozího výsledků umožní přirozenější realizaci operace **INSERT**, i když dosažená složitost je podobná.

Kukaččí hašování pracuje s dvěma tabulkami T_0 a T_1 o velikosti m a s dvěma různými hašovacími funkcemi h_0 a h_1. Řekneme, že množina S je reprezentována funkcemi h_0 a h_1, když

$$S = \{s \mid s \text{ je uložené na nějakém řádku tabulky } T_0 \text{ nebo } T_1\}$$

a pro každé $s \in S$ platí, že s je uložené buď v T_0 na řádku $h_0(s)$ nebo v T_1 na řádku $h_1(s)$, ale nikoliv na obou řádcích. Pak operace **MEMBER**(x) se provede jednoduše, prohlédneme řádek $h_0(x)$ v tabulce T_0 a řádek $h_1(x)$ v tabulce T_1 a $x \in S$, právě když jsme na některém řádku prvek x našli.

Operace **DELETE**(x) je také jednoduchá. Prohlédneme, zda řádek $h_0(x)$ v tabulce T_0 nebo řádek $h_1(x)$ v tabulce T_1 neobsahuje x. Pokud ano, tak ho z této tabulky odstraníme a skončíme, pokud ne, tak hned končíme.

Problém je, kdy existuje reprezentace S. Základní výsledek autorů této metody říká, že když S má jen o málo méně než m prvků, tak pro rozumné funkce je velká pravděpodobnost, že taková dvojice funkcí existuje. Co to jsou rozumné funkce? Zde se opět obrátíme k univerzálním systémům funkcí. Řekneme, že systém funkcí $H = \{h_i \mid i \in I\}$ je c-téměř silně k-univerzální, kde h_i pro $i \in I$ jsou funkce z univerza U do $\{0, 1, \ldots, m-1\}$, když pro každou prostou posloupnost prvků $x_1, x_2, \ldots, x_k$ z univerza U a každou posloupnost prvků $y_1, y_2, \ldots, y_k$ z množiny $\{0, 1, \ldots, m-1\}$ platí

$$\text{Prob}\{i \in I \mid h_i(x_j) = y_j \text{ pro každé } j = 1, 2, \ldots, k\} \leq \frac{c}{m^k}.$$

Když $U = \{0, 1, \ldots, N-1\}$, kde N je prvočíslo a $m = N$, pak polynomy stupně $k+1$ tvoří 1-téměř silně k-univerzální systém. Dále existují konstanty $\varepsilon, c > 0$ takové, že když $|S| = n$, $m \geq (1+\varepsilon)n$ a H je $(1, \lceil c \log n \rceil)$-univerzální, pak když zvolíme $h_0, h_1 \in H$ náhodně a nezávisle, pak pravděpodobnost, že S není reprezentovatelná pomocí h_0 a h_1, je menší než $\frac{1}{n}$. Kukaččí hašování je založeno na tomto výsledku.

Popíšeme operaci **INSERT**(x) za předpokladu, že S je reprezentována pomocí funkcí h_0 a h_1. Když máme vložit prvek a řádek $h_0(x)$ v tabulce T_0 je obsazen prvkem $y_0 \neq x$, pak nahradíme prvek y_0 prvkem x a zkusíme vložit y_0 do tabulky T_1. Navíc si označíme řádek v tabulce T_0. Pak spočítáme $h_1(y_0)$. Pokud řádek $h_1(y_0)$ je volný, pak do něho vložíme prvek y_0 a skončíme. V opačném případě je obsazen prvkem $y_1 \neq y_0$. Když řádek $h_1(y_0)$ není označen, pak y_0 nahradí y_1 na řádku $h_1(y_0)$ v tabulce T_1 a my pokračujeme tak, že označíme řádek $h_1(y_0)$ v tabulce T_1 a pokusíme se vložit y_1 do tabulky T_0. Když je řádek označen, tak končíme s informací, že x nelze vložit do tabulky T_0. Tento proces cyklicky opakujeme.

Důvod, že jsme mohli říct, že x nelze vložit, je ten, že jsme nalezli posloupnost prvků $y_0, y_1, \ldots, y_k$ takovou, že

(1) y_0 je na $h_0(x)$-tém řádku v tabulce T_0;
(2) když $2i+1 < k$, pak y_{2i+1} je na $h_1(y_{2i})$-tém řádku v tabulce T_1;
(3) když $2i+2 < k$, pak y_{2i+2} je na $h_0(y_{2i+1})$-ním řádku v tabulce T_0;
(4) když k je liché, pak pro nějaké $2i+1 < k$ platí $h_0(y_k) = h_0(y_{2i+1})$, když k je sudé, pak pro nějaké $2i < k$ platí $h_0(y_k) = h_0(y_{2i})$.

To znamená, že jsme nalezli posloupnost prvku, která se zacyklila a operace **INSERT** by jí jen posunovala v cyklu. Důsledkem je, že v této reprezentaci nelze x vložit do tabulky T_0. V tom případě vrátíme tabulky do původního stavu. Stejný postup lze provést i s vkládáním prvku x do tabulky T_1.

Na začátku předpokládáme, že není označen žádný řádek. Tuto akci realizuje následují procedura, která má za parametry i a k, kde i nabývá hodnot 0 a 1, a říká, v které tabulce začínáme a k říká, na kterém řádku v tabulce T_i začínáme (předpokládáme, že k-tý řádek v tabulce T_i je obsazen).

Vloz(i, k)
y :=prvek na k-tém řádku tabulky T_i, Q :=prázdná fronta
vlož y do Q a x na k-tý řádek v tabulce T_i a označ ho
$j := i$, $i := 1 - i$, $l := k$, $k := h_i(y)$
while k-tý řádek tabulky T_i je obsazen a není označen **do**
z :=prvek na k-tém řádku tabulky T_i
y vlož na k-tý řádek tabulky T_i a označ ho
$i := 1 - i$, $y := z$, $k := h_i(y)$, vlož y do Q **enddo**
if k-tý řádek tabulky T_i není obsazen **then**
y vlož na k-tý řádek tabulky T_i
zruš označení všech řádků, vyprázdni frontu Q **else**
while Q není prázdná **do**
zruš označení l-tého řádku v tabulce T_j, z :=vrchol fronty Q
odstraň z z Q, z vlož na l-tý řádek tabulky T_j, $j := 1 - j$, $l := h_j(u)$
enddo Výstup: vložení se nepovedlo
endif

Bohužel první cyklus, který hledá volný řádek se může až $|S|$-krát opakovat, kde S je reprezentovaná množina. Aby toto nemohlo nastat (i když tento jev není moc pravděpodobný), tak je stanovená hodnota q závislá na velikosti reprezentované množiny, která omezuje počet opakování tohoto cyklu a tedy podprocedura vyžaduje čas $O(q)$.

Nyní popíšeme vlastní algoritmus **INSERT**(x). Spočítáme $h_0(x)$ a $h_1(x)$. Pokud $h_0(x)$-tý řádek tabulky T_0 nebo $h_1(x)$-tý řádek tabulky T_1 obsahuje x, tak končíme. Pokud $h_0(x)$-tý tabulky T_0 je prázdný, tak tam vložíme x a končíme. Když není prázdný, tak testujeme, zda $h_1(x)$-tý řádek tabulky T_1 je prázdný, v tom případě tam vložíme x a končíme. Když oba řádky jsou neprázdné a neobsahují x, pak zavoláme podproceduru **Vlož**$(0, h_0(x))$, a když úspěšně přerovná prvky, tak končíme. Když dostaneme zprávu "vložení se nepovedlo", tak zavoláme podproceduru **Vlož**$(1, h_1(x))$. Když se jí povede přerovnat prvky, tak končíme. Když obě volání podprocedury **Vlož** končí hláškou, "vložení se nepovedlo", tak oznámíme, že **INSERT**(x) nelze provést, reprezentace vložené množiny nelze rozšířit o prvek x. Tady je formální zápis algoritmu.

INSERT(x)
Spočítej $k_0 := h_0(x)$ a $k_1 := h_1(x)$
if x je na k_0-tém řádku tabulky T_0 nebo na k_1-ním řádku tabulky T_1 **then** stop **endif**
if k_0-tý řádek tabulky T_0 je prázdný **then**
vlož x na k_0-tý řádek tabulky T_0, stop **endif**
if k_1-ní řádek tabulky T_1 je prázdný **then**
vlož x na k_1-ní řádek tabulky T_1, stop
endif Vlož$(0, k_0)$
if podprocedura **Vlož** hlásí, že vnoření se nepovedlo **then**
Vlož$(1, k_1)$
if podprocedura **Vlož** hlásí, že vnoření se nepovedlo **then**
Výstup: prvek x nelze do této struktury vložit
endif endif

Pokud dostaneme hlášení, že současná reprezentace reprezentované množiny neumožňuje její rozšíření o prvek x, pak provedeme přehašování. To znamená, že zvolíme náhodně a nezávisle dvě funkce v systému H a hledáme reprezentaci $S \cup \{x\}$ pomocí nových dvou funkcí tak, že opakujeme algoritmus operace **INSERT**(y) pro $y \in S \cup \{x\}$ (S je reprezentovaná množina).

Všimněme si, že nalezení navštíveného vrcholu znamená, že prvky uložené pomocí h_0 a h_1 se "zacyklily". Pokud se to stane pro tabulku T_0 i T_1, tak to znamená, že reprezentace $S \cup \{x\}$ pomocí h_0 a h_1 neexistuje.

Je třeba ještě hlídat velikosti reprezentovaných množin. Když po **INSERT**(x) je v množině S mnoho prvků nebo po operaci **DELETE**(x) je málo prvků, tak se tabulky zdvojnásobí nebo se stanou poloviční.

Autoři ukázali, že když H je 1-téměř silně $\lceil \log n \rceil$-univerzální systém, pak operace **INSERT** a **DELETE** mají očekávaný amortizovaný čas konstantní.

Kukaččí hašování navrhli a analyzovali R. Pagh a F. F. Rodler v publikaci z roku 2004.