Text doplněných přednášek.

Binární vyvážené vyhledávací stromy.
V minulém semestru jsme si ukazovali vztah červeno-černých stromů a $(2,4)$-stromů a jejich možné zobecnění pro (a,b)-stromy. Nyní uvedeme jinou kombinatorickou charakteristiku červeno-černých stromů. Pomocí ní ukážeme vztah červeno-černých stromů a AVL-stromů. (Předpokládáme definici červeno-černých stromů, kde listy reprezentují intervaly).

Mějme úplný binární strom T, pak funkce f z vrcholů úplného binárního stromu T do přirozených čísel splňující podmínky

(r1) když x je list, pak $f(x) = 0$ a $f(\text{otec}(x)) = 1$;
(r2) pro každý vnitřní vrchol v různý od kořene platí $f(v) \leq f(\text{otec}(v)) \leq f(v)+1$;
(r3) pro každý vnitřní vrchol v takový, že v i otec(v) jsou různé od kořene, platí $f(\text{ded}(v)) > f(v)$

se nazývá hodnostní funkce T (anglicky rank of T).

Věta 1. *Úplný binární strom T je červeno-černý strom, právě když má hodnostní funkci.*

Důkaz. Předpokládejme, že T je červeno-černý strom. Definujme funkci f na vrcholech stromu T tak, že pro každý vrchol v je $f(v)$ počet černých vrcholů na některé cestě z některého syna vrcholu v do listu. Ukážeme, že f je korektně definovaná hodnostní funkce. Podstrom určený vrcholem v je červeno-černý strom, a tedy všechny cesty z vrcholu v do listů mají stejný počet černých vrcholů. Proto mají všechny cesty z některého syna vrcholu v do listů stejný počet černých vrcholů. Odtud plyne korektnost definice f, protože nezávisí ani na volbě syna ani na volbě cesty.

Dokážeme, že f je hodnostní funkce. Z definice f okamžitě plyne, že f splňuje (r1). Protože cesta z vrcholu v do listu má nejvýše o jeden černý vrchol víc než cesta z jeho syna, dostáváme, že $f(v) \leq f(\text{otec}(v)) \leq f(v)+1$, a tedy je splněna podmínka (r2). Navíc $f(v) < f(\text{otec}(v))$, právě když v je obarven černě. Nyní podmínka (r3) plyne z toho, že červeně obarvený vrchol je buď kořen nebo jeho otec je černě obarvený, a tedy f je hodnostní funkce T.

Nyní předpokládejme, že úplný binární strom T má hodnostní funkci f. Vrchol v stromu T různý od kořene obarvíme černě, když $f(v) < f(\text{otec}(v))$, a obarvíme ho červeně, když $f(v) = f(\text{otec}(v))$. Když $f(r) = f(v)$ pro kořen r stromu T a některého jeho syna v, pak obarvíme r černě, jinak obarvíme r libovolně. Zřejmě každý vrchol je obarven právě jednou barvou. Z podmínky (r1) okamžitě plyne, že listy jsou obarveny černě. Mějme červeně obarvený vrchol v stromu T, který není kořen. Pak $f(v) = f(\text{otec}(v))$. Když otec(v) není kořen, pak podle (r2) $f(\text{ded}(v)) > f(v) = f(\text{otec}(v))$, a tedy otec$(v)$ je obarven černě. Když otec(v) je kořen, pak podle definice je otec(v) obarven černě a tedy platí podmínka pro červeně obarvené vrcholy.

Protože f je funkce do přirozených čísel taková, že

$f(v) = 0$ pro každý list v;
vrchol v různý od kořene je obarven černě, právě když $f(v)+1 = f(\text{otec}(v))$;

vrchol v různý od kořene je obarven červeně, právě když $f(v) = f(\text{otec}(v))$, dostáváme, že každá cesta z vrcholu v různého od kořene do některého listu obsahuje $f(\text{otec}(v))$ černých vrcholů. Speciálně, každá cesta z každého syna kořene do některého listu obsahuje f(kořen stromu) černých vrcholů. Z tohoto plyne, že všechny cesty z kořene do listu mají stejný počet čern ych vrcholů. Tedy T je červeno-černý strom. □

Snaha nalézt lepší algoritmy pro operaci **DELETE** vedla k zavedení polovyvážených stromů. Autor těchto stromů (Olivié, 1980) popsal algoritmy pro operace **INSERT** a **DELETE**, které vyžadovaly konstantní počet rotací a dvojitých rotací. Na tento výsledek navázal Tarjan 1983, který nalezl algoritmus pro červeno-černé stromy vyžadující nejvýše jednu rotaci a jednu dvojitou rotaci nebo dvě rotace (byl ukázán v minulém semestru). Také ukázal, že strom je červeno-černý, právě když je polovyvážený. Řekneme, že úplný binární strom T je polo-vyvážený, když $v(v) \leq 2s(v)$ pro každý vrchol v stromu T, kde
$v(v)$ je délka nejdelší cesty z vrcholu v do listu v podstromu vrcholu v – tj. $v(v)$ je výška vrcholu v;
$s(v)$ je délka nejkratší cesty z vrcholu v do listu v podstromu vrcholu v.

Věta 2. *Strom T je červeno-černý, právě když je polovyvážený.*

Důkaz. Nechť T je červeno-černý strom. Vezměme hodnostní funkci f stromu T definovanou ve Větě 1. Pak $f(v)$ je počet černých vrcholů na některé cestě z některého syna vrcholu v do listu v podstromu tohoto syna. Mějme cestu P z vrcholu v do do listu v jeho podstromu a nechť k je počet černých vrcholů na cestě $P \setminus \{v\}$. Pak délka cesty P je alespoň k a nejvýše $2k$. Tedy pro každý vrchol v stromu T platí

$$f(v) - 1 \leq s(v) \leq v(v) \leq 2f(v) - 1.$$

Proto T je polovyvážený.

Nechť naopak T je polovyvážený strom. Budeme definovat indukcí funkci f z vrcholů stromu T do přirozených čísel tak, aby pro každý vrchol v stromu T platilo $\lceil \frac{v(v)}{2} \rceil \leq f(v) \leq s(v)$ (z $v(v) \leq 2s(v)$ plyne $\lceil \frac{v(v)}{2} \rceil \leq s(v)$). Začeneme u kořene. Když r je kořen, zvolme $f(r)$ jako přirozené číslo takové, že $\lceil \frac{v(r)}{2} \rceil \leq f(r) \leq s(r)$. Mějme definovanou funkci $f(v)$ pro vnitřní vrchol v stromu T tak, že platí $\lceil \frac{v(v)}{2} \rceil \leq f(v) \leq s(v)$, a nechť w je syn vrcholu v. Položme $f(w) = \max\{f(v) - 1, \lceil \frac{v(w)}{2} \rceil\}$. Pak $\lceil \frac{v(w)}{2} \rceil \leq f(w)$. Z $f(v) \leq s(v) \leq s(w) + 1$ dostáváme, že $f(v) - 1 \leq s(w)$. Protože $\lceil \frac{v(w)}{2} \rceil \leq s(w)$, platí $\lceil \frac{v(w)}{2} \rceil \leq f(w) \leq s(w)$. Navíc můžeme říct, že $f(v) \leq f(w) + 1$. Protože $v(w) + 1 \leq v(v)$ a $\lceil \frac{v(v)}{2} \rceil \leq f(v)$, dostáváme $\lceil \frac{v(w)}{2} \rceil \leq f(v)$, a proto $f(w) \leq f(v)$. Tedy $f(w) \leq f(v) \leq f(w) + 1$. Protože w byl libovolný syn vrcholu v, je f definována pro oba syny vrcholu v a indukcí dostáváme, že f je definována pro všechny vrcholy stromu T a splňuje podmínku (r2).

Protože pro list l stromu T platí $v(l) = s(l) = 0$, dostáváme, že $f(l) = 0$. Když v je otec listu, pak $s(v) = 1 \leq v(v) \leq 2s(v) = 2$, a proto $\lceil \frac{v(v)}{2} \rceil = 1 = f(v)$ a f splňuje (r1). Mějme vrchol v stromu T takový, že v ani otec(v) nejsou kořeny stromu T. Označme $t = \text{otec}(v)$ a $u = \text{ded}(v)$. Z (r2) plyne $f(v) \leq f(t) \leq f(u)$. Protože $f(v) = \max\{f(t) - 1, \lceil \frac{v(v)}{2} \rceil\}$, tak platí buď $f(v) < f(t) \leq f(u)$ nebo $\lceil \frac{v(v)}{2} \rceil > f(t) - 1 \geq \lceil \frac{v(t)}{2} \rceil - 1$. Předpokládejme, že $\lceil \frac{v(v)}{2} \rceil > f(t) - 1 \geq \lceil \frac{v(t)}{2} \rceil - 1$.

Pak $\lceil \frac{v(v)}{2} \rceil + 1 > \lceil \frac{v(t)}{2} \rceil$ a z $v(t) \geq v(v) + 1$ pak plyne $\lceil \frac{v(t)}{2} \rceil \geq \lceil \frac{v(v)}{2} \rceil$, a tedy $\lceil \frac{v(v)}{2} \rceil = \lceil \frac{v(t)}{2} \rceil$. Proto $v(t) = v(v) + 1$ je sudé a $f(v) = f(t) = \lceil \frac{v(t)}{2} \rceil$. Pak $v(u) \geq v(t) + 1$ implikuje, že $f(v) = f(t) = \lceil \frac{v(t)}{2} \rceil < \lceil \frac{v(u)}{2} \rceil \leq f(u)$. Tedy f splňuje podmínku (r3), takže je to hodnostní funkce pro T a podle Věty 1 T je červeno-černý strom. □

Dále ukážeme vztah mezi AVL-stromy a červeno-černými stromy.

Věta 3. *Každý AVL-strom je červeno-černý strom.*

Důkaz. Nechť T je AVL-strom. Pak pro každý vnitřní vrchol v stromu T platí, že $|v(u) - v(w)| \leq 1$, kde u je levý syn vrcholu v a w pravý syn vrcholu v. Dokážeme, že když $v(u) \leq 2s(u)$ a $v(w) \leq 2s(w)$, pak $v(v) \leq 2s(v)$. Platí $s(v) = 1 + \min\{s(u), s(w)\}$ a $v(v) = 1 + \max\{v(u), v(w)\}$. Pak

$$2s(v) = 2 + 2\min\{s(u), s(w)\} \geq 2 + \min\{v(u), v(w)\}.$$

Z $|v(u) - v(w)| \leq 1$ plyne, že

$$\max\{v(u), v(w)\} - \min\{v(u), v(w)\} \leq 1,$$

a proto $2s(v) \geq 2 + \min\{v(u), v(w)\} \geq 1 + \max\{v(u), v(w)\} = v(v)$. Protože pro každý list l stromu T platí $0 = v(l) = 2s(l)$, dostáváme indukcí, že pro každý vrchol v stromu T platí $v(v) \leq 2s(v)$, a tedy T je polovyvážený strom a z Věty 2 plyne požadované tvrzení. □

Obecné vyvážené binární vyhledávací stromy.

Tyto vztahy a výsledky a zjištění, že řada velkých softwarových firem se nedostala dál než ke spojovým seznamům, a použití vyvážených stromů by značně zrychlilo jejich produkty, vedly Anderssona k definici obecně vyvážených binárních vyhledávacích stromů. Zjistil, že červeno-černé stromy a AVL-stromy jsou příliš složité pro programátory těchto firem (efektivní programy pro tyto struktury lze nalézt na Internetu). Proto navrhl následující strukturu.

Množina $S \subseteq U$ je reprezentována binarním vyhledávacím stromem a navíc jsou dány horní hranice pro hloubku stromu a počet operací **DELETE** provedených od posledního vyvažování. Hloubka stromu je vždy shora omezena hodnotou $c \log |S|$ pro vhodné $c > 1$ (to je podmínka obecné vyváženosti). Autor experimentálně ověřoval chování datové struktury pro $c \approx 1.3$ (pak hloubka těchto stromů v nejhorším případě je menší než hloubka v nejhorším případě u červeno-černých stromů a u AVL-stromů).

Operace **MEMBE**R(x) a **DELETE**(x) se provedou stejným způsobem jako v klasických (nevyvážených) binárních vyhledávacích stromech. Po úspěšném provedení operace **DELETE** se počet operací **DELETE** zvětší o 1 a když překročí stanovenou hranici, provede se vyvážení celého stromu reprezentujícího množinu S. Operace **INSERT**(x) se také provede jako v klasických binárních vyhledávacích stromech, ale navíc při ní počítáme délku cesty z kořene do listu, který bude reprezentovat x. Po přidání x ji zvětšíme o 1 a když překročí výšku stromu, zvětšíme výšku stromu. Pokud výška překročí stanovené omezení na výšku stromu (toto omezení závisí na velikosti reprezentované množiny), pak se provede vyvážení stromu.

Vyvažování stromu se provádí tak, že se najde vrchol v, jehož podstrom se má vyvážit, a tento podstrom se nahradí podstromem reprezentujícím stejnou množinu, ale s nejmenší výškou. Při operaci **INSERT**(x) vrcholem v bude vrchol s nejmenší výškou takový, že jeho podstrom není vyvážený (tj. nesplňuje omezení na svou výšku vzhledem k velikosti jim reprezentované množiny). Nahrazením tohoto podstromu podstromem s nejmenší výškou se zmenší výška celého stromu a ten pak bude splňovat omezení dané na výšku stromu (protože před operací toto omezení splňoval). Při vyvažování po operaci **DELETE** provedeme vyvažování pro kořen stromu, tj. nahradíme původní strom novým binárním vyhledávacím stromem, který reprezentuje stejnou množinu, ale má nejmenší výšku.

Abychom mohli realizovat tuto operaci, je potřeba znát v každém vrcholu velikost množiny reprezentované podstromem tohoto vrcholu. Proto je potřeba po úspěšné operaci **INSERT** nebo **DELETE** (tj. když se přidal nebo odebral prvek) projít cestu od upravovaného vrcholu nazpět ke kořeni a aktualizovat velikost množin reprezentovaných podstromy na této cestě.

Anderssonem navrhované vyvažování nejprve převede nevyvážený podstrom pomocí rotací a dvojitých rotací do úplně zdegenerovaného tvaru (tj. je to jediná cesta z kořene, k níž jsou přidány listy) a pak opět pomocí rotací a dvojitých rotací vytvořit binární vyhledávací strom s nejmenší výškou. V literatuře lze nalézt celou řadu efektivnějších způsobů pro konstrukci binárního vyhledávacího stromu s nejmenší výškou, ale nikde jsem nenašel vzájemné porovnání jejich efektivity. Proto lze říct, že obecné vyvážené binární vyhledávací stromy otevírají celou řadu problémů, které by bylo dobré vyřešit.

Relaxované stromy.

Nyní popíšeme jinou strategii pro práci s vyváženou stromovou datovou strukturou. Tato strategie umožňuje současnou práci více uživatelů a je vhodná i pro práci v dávkovém režimu. S ideou, na které je založena, jsme se poprvé setkali při líné implementaci binomiálních hald. Je založena na myšlence, že odložení vyvažování na pozdější dobu vede k vzájemnému vyrušení některých vyvažovacích požadavků. Nevýhodou této ideje je fakt, že ztrátou vyváženosti se může výrazně prodloužit čas potřebný k vyhledávání (logaritmický vyhledávací čas může vzrůst až na lineární). Na druhé straně víme, že při rovnoměrném rozdělení dat je tento nárůst nepravděpodobný. V praxi se sice s rovnoměrným rozdělením dat setkáváme málokdy, ale může to platit i pro rozdělení, která jsou mu blízká, což jsou mnohá rozdělení z praxe.

Metoda od sebe odděluje vyhledávací a vyvažovací operace. Místo toho, aby se provedlo vyvažování, se jen zaznamená požadavek na vyvažování do fronty požadavků. Výhoda této metody se zvláště projevuje v časových špičkách, kdy přichází mnoho požadavků od uživatelů a nelze stihnout vyvažování, nebo při práci v dávkovém režimu (např. když administrátor odeslal dávku požadavků, aby odstranil část datové struktury poškozenou výpadkem proudu). Další výhoda této metody spočívá ve faktu, že ji lze snadno modifikovat pro všechny běžně používané vyvážené stromy. Použití vyžaduje ošetření dvou problémů. O prvním jsme se již zmínili, je to možný nárůst času pro vyhledávání způsobený degenerací datové struktury. Druhý problém je, zda lze jednoduše vyvážit binární strom, který vznikl několika aktualizačními operacemi (bez následného vyvažování) z vyváženého binárního vyhledávacího stromu (bez nového budování celého vyváženého vyhledáva-

cího stromu, jen na základě reprezentovaných dat). S ním je spojena otázka strategie řešení vyvažovacích požadavků.

Stromy vzniklé touto metodou se nazývají relaxované. Přitom konkrétních realizací této základní ideje je pro každý typ vyvážených stromů několik (záleží na tom, který parametr je preferován). Popíšeme jeden relaxovaný model pro červeno-černé stromy.

Uvažovaný model má data uložená v binárním vyhledávacím stromu, jehož vnitřní vrcholy jsou obarveny buď červeně nebo černě (obarvení oběma barvami není přípustné) a listy jsou obarveny černě. Navíc je dán soubor vyvažovacích požadavků (budeme ho nazývat fronta vyvažovacích požadavků) a pokud je tento soubor prázdný, pak strom reprezentující data je vyvážený červeno-černý strom. Nad daty pracuje současně více procesů, které jsou dvou typů:

uživatelský – provádí pouze vyhledávání, přidávání a ubírání prvků a když po aktualizaci vznikne požadavek na vyvažování (význam i formu upřesníme později), dá tento požadavek do fronty vyvažovacích požadavků.
správcovský – bere vhodné požadavky z fronty vyvažovacích požadavků a provádí je. Může se stát, že buď daný požadavek úplně ošetří nebo ho transformuje v jiný požadavek bližší ke kořeni stromu. Tím se postupně odstraňuje nevyváženost stromu.

Ideálem je v jistých periodách vyprázdnit frontu požadavků a tím vytvořit klasický červeno-černý strom. Počet pracujících správcovských procesů není fixní a závisí na délce fronty vyvažovacích požadavků.

Nyní popíšeme sémantiku a formu požadavků. Budeme mít požadavky dvou typů – b a v. Když je s vrcholem v svázán požadavek b, pak s ním už není svázán žádný další požadavek (tj. požadavek b je neslučitelný s každým dalším požadavkem, zatímco požadavků typu v na vrchol v může být víc). Pokud je s vrcholem svázán nějaký požadavek, pak vrchol má ukazatel na tento požadavek ve frontě vyvažovacích požadavků. Požadavek b na vrchol v znamená, že v je červený a má červeného otce (to platí v okamžiku vznesení požadavku, tato situace se později může změnit). Požadavek v na vrchol v znamená, že v je černý a v cestách z kořene stromu do listů procházejících vrcholem v chybí jeden černý vrchol (proti ideální cestě). Z toho plyne, že když je na vrchol v vloženo k požadavků v, pak v cestách z kořene do listů procházejících vrcholem v chybí k černých vrcholů, tj. kdyby každý vrchol v s k požadavky v představoval $k+1$ černých vrcholů, pak všechny cesty z kořene do listů by měly stejný počet černých vrcholů).

Ve frontě vyvažovacích požadavků je každý požadavek specifikován svým typem a ukazatelem na vrchol, kde vznikl.

Neformálně popíšeme práci jednotlivých procesů. Uživatelský proces provádí jednu ze tří operací: **MEMBER**, **INSERT** a **DELETE**.

Operace **MEMBER**(x) klasickým vyhledáváním zjistí, zda x patří do reprezentované množiny, a oznámí to uživateli (pokud patří, oznámí také adresu dat spojených s prvkem x).

Operace **INSERT**(x) klasickým vyhledáváním zjistí, zda x patří do reprezentované množiny. Když patří, operace končí (zde je několik přirozených alternativ – např. oznámí neúspěšné vložení prvku nebo oznámí adresu dat spojených s prvkem x

atd.) Když x nepatří do reprezentované množiny, tak vyhledávání skončilo v listu t (který reprezentuje interval obsahující x). Nyní změní list t na vnitřní vrchol, vytvoří dva černé syny vrcholu t, které budou listy, uloží data spojená s x a jejich adresu spolu s prvkem x spojí s vrcholem t. Když vrchol t vznášel požadavek v, pak smaže jeden požadavek v vznesený vrcholem t a nechá vrchol t černý, když vrchol t nevznášel žádný požadavek v, tak změní jeho barvu na červenou a když otec vrcholu t je červený, vytvoří pro vrchol t požadavek b a vloží ho do fronty vyvažovacích požadavků (propojí vrchol t a tento požadavek ukazateli).

Operace **DELETE**(x) klasickým způsobem zjistí, zda x patří do reprezentované množiny. Když nepatří, operace končí (případně oznámí neúspěch). V opačném případě nalezne vrchol t a jeho syna l, které mají být odstraněny, uvolní data spojená s prvkem x. Když t nereprezentoval x, pak data spojená s t přesune na vrchol reprezentující x. Pak odstraní vrchol t a list l a na místo vrcholu t dá bratra listu l, kterého obarví na černo. Když t i bratr l byly obarveny černě, vytvoří požadavek v spojený s bratrem l a vloží ho do fronty vyvažovacích požadavků (a propojí ukazateli bratra l s tímto požadavkem). Požadavky b spojené s vrcholy t a bratrem l se ruší a odstraní se z fronty vyvažovacích požadavků (když t nebo bratr l měly požadavek b, pak jejich odstraněním žádný nový požadavek nevzniká). Navíc bratr l dědí všechny požadavky v spojené s vrcholem t (tím se mohou hromadit požadavky v spojené s jedním vrcholem).

Nyní neformálně popíšeme práci správcovského procesu. Proces provede jednu z následujících akcí:

Zruší (a odstraní z fronty) jakýkoliv požadavek na kořen stromu.

Zruší (a odstraní z fronty) požadavek b na vrchol v, když otec vrcholu v je černý.

Když je na vrchol v vloženo i požadavků v a na bratra vrcholu v je vloženo j požadavků v, pak zruší $\min(i, j)$ požadavků v na vrcholy v a bratr(v). Když otec vrcholu v je černý, pak vloží nových $\min(i, j)$ požadavků v na otce vrcholu v. Když otec vrcholu v je červený, pak změní jeho barvu na černou a vloží na něho $\min(i, j) - 1$ požadavků v.

Když je na vrchol v vložen požadavek b a jeho otec je červený a je kořen stromu, pak obarvíme otce vrcholu v na černo a požadavek zrušíme.

Když je na vrchol v vložen požadavek b, otec vrcholu v je červený, děd vrcholu v je černý, bratr u otce vrcholu v je červený, pak obarvíme otce vrcholu v a vrchol u na černo a zrušíme požadavek na vrchol v. Když na děda vrcholu v je vloženo i požadavků v pro $i > 0$, pak odstraníme jeden požadavek v na děda vrcholu v. Pokud na děda vrcholu v není vložen žádný požadavek v (požadavek b na něho nemůže být vložen, protože je černý), obarvíme ho na červeno a pokud otec děda vrcholu v je také červený, pak vložíme na děda vrcholu v požadavek b. Když na vrchol u je vložen požadavek b, tak ho zrušíme.

Když na vrchol v je vložen požadavek b, otec z vrcholu v je červený, děd w vrcholu v je černý, bratr otce vrcholu v je černý a v není lomený, pak provedeme rotaci vrcholů z a w, w obarvíme červeně, z obarvíme černě, vrchol z zdědí všechny požadavky v vrcholu w a zrušíme požadavek b na vrchol v.

Když na vrchol v je vložen požadavek b, otec z vrcholu v je červený, děd w vrcholu v je černý, bratr otce vrcholu v je černý a v je lomený, pak provedeme dvojitou rotaci vrcholů v, z a w, w obarvíme červeně, v obarvíme černě, zrušíme požadavek b na vrchol v a vrchol v zdědí všechny požadavky v na vrchol w.

Srovnejte tyto tři akce s vyvažováním po operaci **INSERT** v klasickém červeno-černém stromu.

Když na vrchol v je vloženo j požadavků v pro $j > 0$, bratr u vrcholu v je černý a není na něho vložen žádný požadavek, synové vrcholu u jsou černí, pak obarvíme vrchol u červeně, zrušíme jeden požadavek v na vrchol v a když byl otec vrcholu v červený, tak ho obarvíme na černo, a pokud byl černý, tak vytvoříme požadavek v na otce vrcholu v a vložíme ho do fronty vyvažovacích požadavků.

Když na vrchol v je vloženo j požadavků v pro $j > 0$, bratr u vrcholu v je černý a není na něho vložen žádný požadavek, syn w vrcholu u, který je lomený, je červený, pak provedeme dvojitou rotaci na vrcholy w, u a otec(v), obarvíme vrchol otec(v) černě, zrušíme jeden požadavek v na vrchol v a vrchol w zdědí barvu i požadavky spojené s vrcholem otec(v). Zrušíme také případný požadavek b na vrchol w.

Když na vrchol v je vloženo j požadavků v pro $j > 0$, bratr u vrcholu v je černý a není na něho vložen žádný požadavek, syn w vrcholu u, který není lomený, je červený, pak provedeme rotaci na vrcholy u a otec(v), obarvíme vrcholy otec(v) a w černě, zrušíme jeden požadavek v na vrchol v a vrchol u zdědí barvu i požadavky spojené s vrcholem otec(v). Zrušíme také případný požadavek b na vrchol w.

Když na vrchol v je vloženo j požadavků v pro $j > 0$, a bratr u vrcholu v je červený a není na něho vložen žádný požadavek, pak provedeme rotaci na vrcholy u a otec(v)z, obarvíme otec(v) na červeno (a nebude na něm žádný požadavek), vrchol u obarvíme na černo a zdědí všechny požadavky vrcholu otec(v).

Předchozí akce srovnejte s operací **DELETE** pro klasické červeno-černé stromy.

Nyní stručně popíšeme práci s frontou vyvažovacích požadavků. Správcovský proces se náhodně rozhodne, zda chce odstranit požadavky na kořen, nebo zda chce odstranit požadavek typu b, nebo zda chce odstranit požadavek typu v.

Když chce odstranit požadavek na kořen, pak prohledáním fronty vyvažovacích požadavků nalezne požadavek na kořen. Zablokuje kořen. Během této akce nesmí být kořen argumentem žádné rotace nebo dvojité rotace (pak by přestal být kořenem a požadavek nejde odstranit). Požadavek odstraní a kořen uvolní. Místo prohledávání fronty je rychlejší začít u kořene, zjistit, zda je na něj položen požadavek, a nalézt tento požadavek pomocí ukazatele.

Když chce odstranit požadavek typu b, pak nejprve nalezne vrchol v, který není kořen, je na něho položen požadavek b a na otce vrcholu v není položen požadavek b. Zablokuje vrchol v a testuje, zda otec vrcholu není černý nebo není kořen stromu. Když otec vrcholu v je černý nebo je kořen stromu, pak zablokuje otce vrcholu v a provede akci. Po jejím provedení uvolní oba vrcholy. Když otec vrcholu v je červený a není kořen stromu a děd vrcholu v je černý, tak zablokuje otce vrcholu v, bratra otce vrcholu v a děda vrcholu v a podle vlastností bratra otce vrcholu v provede akci a vrcholy uvolní.

Když chce odstranit požadavek v, nalezne vrchol v, na kterého je vložen požadavek v a není kořen a na jeho bratru je také vložen požadavek v, pak tyto požadavky posune směrem ke kořeni tak, že na vrchol v nebo na jeho bratra nenívložen žádný požadavek v.

Když chce odstranit požadavek v, tak nalezne vrchol v, na který je vložen požadavek v a není kořen a na bratra vrcholu v není vložen požadavek b. Pak zablokuje otce

vrcholu v, bratra vrcholu v i vrchol v. Podle vlastností bratra vrcholu v provede akci. V případě, že bratr vrcholu v je černý a není na něho vložen žádný požadavek, tak zablokuje i syny bratra vrcholu v. Po provedení akce uvolní vrcholy. Pokud bratr vrcholu v byl obarven červeně, tak provede dvě akce na v rchol v, a pak teprve uvolní vrcholy.

Při volbě, jaký požadavek bude ošetřovat správcovský server, by největší prioritu mělo mít ošetřování kořene, pak ošetření požadavku b a pak požadavku v. To by se mělo odrazit při volbě akce správcovského procesu. Vhodný poměr priorit není znám. Zdá se, že je také výhodné začít hledat požadavky ve stromě a pak přejít do fronty vyvažovacích požadavků pomocí ukazatele. Zablokovaný vrchol znamená, že není přístupný jinému správcovskému procesu. Jen při provádění rotace nebo dvojité rotace jsou její argumenty zablokovány i pro uživatelské procesy.

Následující tvrzení se ověří přímo (viz klasické červeno-černé stromy).

Věta 4. *Když fronta vyvažovacích požadavků je neprázdná, tak v ní vždy existuje požadavek, který může obsloužit správcovský proces. Když vrchol v binárního vyhledávacího stromu reprezentujícího data je červený a jeho otec je také červený, pak na vrchol v byl vložen požadavek b. V každém okamžiku platí, že když každý vrchol binárního vyhledávacího stromu reprezentujícího data, na který je vloženo i požadavků* v *pro* $i > 0$*, je nahrazen* $i+1$ *černými vrcholy, pak všechny cesty z kořene do listů mají stejný počet černých vrcholů.*

Z této věty plyne, že když je fronta požadavků prázdná, pak binární vyhledávací strom reprezentující data je červeno-černý. Vzniká však otázka, zda každá posloupnost akcí správcovských procesů vede k vyprázdnění fronty požadavků. Ukážeme to pomocí amortizované složitosti. Požadavek b na vrchol v v hloubce i a požadavek v na vrchol v v hloubce i, jehož otec je červený, ohodnotíme i, požadavek v na vrchol v v hloubce i, jehož otec je černý, ohodnotíme $i + 2$, a ocenění fronty požadavků je součet ohodnocení jednotlivých požadavků. Pak platí

Tvrzení 5. *Ohodnocení každé fronty vyvažovacích požadavků je nezáporné a je* 0*, právě když je fronta prázdná. Každá akce správcovského procesu snižuje ocenění fronty vyvažovacích požadavků.*

Předchozí tvrzení ukazuje, že každá posloupnost akcí správcovského procesu vede k vyprázdnění fronty vyvažovacích požadavků, a je zřejmé, že posloupnost operací v klasických červeno-černých stromech vyžaduje více vyvažovacích akcí. Jsou zde otevřené otázky:

Lze najít optimální strategii pro správcovské procesy? S jakou pravděpodobností je binární strom v závislosti na délce fronty ještě blízký pravidelnému úplnému binárnímu stromu?

Podobné modely jsou studovány i pro AVL-stromy a (a, b)-stromy.

Dvě volby.

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 je 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 6. *Když Z je binomická náhodná proměnná s parametry n a p, pak*

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

Lemma 7. *Mějme posloupnost náhodných proměnných $X_1, X_2, \ldots, X_n$ a binárních náhodných proměnných $Y_1, Y_2, \ldots, Y_n$ takových, že $Y_i = Y_i(X_1, X_2, \ldots, X_i)$ a*

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

pro každé i, pak

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

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

Proof. Pro každé i uvažujme indikátor I_i takový, že $I_i = 1$ právě když $Y_i = 1$. Pak tento indikátor I_i je Bernoulliovský pokus s pravděpodobností p a platí, že $\text{Prob}(I_i = 1)$ majorizuje $\text{Prob}(Y_i = 1)$. Tedy jednoduchou indukcí dostaneme

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

Teď si stačí uvědomit, že součet n Bernoulliovských pokusů s pravděpodobností p dává binomickou náhodnou proměnnou s 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 pravděpodobností" 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 a $\mu_i(t)$ 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 $\nu_i(t) \leq \mu_i(t)$.

Když β_i je 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 již z Markovovy nerovnosti plyne, že $\beta_{i+1} \leq 2n\left(\frac{\beta_i}{n}\right)^d$ s pravděpodobností větší než $\frac{1}{2}$. Proto 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 moc předpokládat, že s velkou pravděpodobností platí $\nu_i = \nu_i(n) \leq \beta_i$ pro všechna 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.

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 7 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 $\sum_{t=1}^{n} Y_t = \mu_{i+1}$ a $\nu_{i+1} \leq \mu_{i+1}$, 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 6 pro $k = \beta_{i+1} = 2np_i$ a dostaneme

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

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

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

protože $\mathrm{Prob}(\neg\mathcal{E}_4) = 0$.
Dále indukcí podle i ukážeme, že

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

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

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

kde první rovnost je definice $\beta_{(i+1)+4}$ druhá rovnost využívá indukčního předpokladu.
Z tohoto výrazu dostáváme, že $\beta_{i+4} \leq \frac{n}{2^{d^i}}$. Z definice $p_i = \frac{\beta_i^d}{n^d}$ plyne, 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 6 dostaneme, že

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

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, dostaneme

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

Zřejmě platí

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

Dále platí

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

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 při hrádce mají pozici aspoň i^*+2 je $\left(\frac{12\ln n}{n}\right)^{2d}$.
Spojením těchto odhadů dostaneme

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

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

Věta 8. *Předpokládejme, že máme n přihrádek a n míčků a že $d \geq 2$ 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

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

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 $\gamma_i \geq \frac{n}{2^{10d^{i-1}}}$. 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 $\{0, 1, \ldots, m-1\}$ tak, že když vybereme n prvků pro $n \leq m$, pak získáme nezávislé d-tice s rovnoměrným rozdělením a prvky v d-tice jsou navzájem nezávislé. Navíc musíme umět tyto funkce rychle vyčíslit. Toto splnit je obtížné, ale výsledek mimo jiné motivoval definici kukaččího hašování, které spojilo tuto ideu s perfektním hašováním.
Abychom získali d-tice navzájem nezávislých čísel budeme modifikovat definici univerzálního hašování. Druhý problém, aby byly d-tice s rovnoměrným rozložením neumíme vyřešit jinak než předpokladem náhodného výběru prvků.

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

$$|\{i_1, i_2, \ldots, i_k\} \mid \forall j = 1, 2, \ldots, k,\ i_j \in I,\ h_{i_j}(x) = y_j\}| = \frac{|I|^k}{m^k}.$$

a navíc pro každé $x \in U$ a $y \in \{0, 1, \ldots, m-1\}$ platí

$$|\{i \in I \mid f_i(x) = y\}| = \frac{|I|}{m}.$$

Pokud máme nezávislý k-univerzální soubor funkcí $\mathcal{H}$ z univerza do množiny $\{0, 1, \ldots, m-1\}$ a náhodně a nezávisle vybereme k funkcí $f_1, f_2, \ldots, f_k$ ze souboru $\mathcal{H}$, pak pro každé $x \in U$ a každou k-tici prvků $\{y_i\}_{i=1}^k$z množiny $\{0, 1, \ldots, m-1\}$ platí

$$\text{Prob}\{f_i(x) = y_i \mid \forall i = 1, 2, \ldots, k\} = \frac{1}{m^k}$$

(za předpokladu, že funkce je určena indexem). Protože pro každé $x \in U$ a každé $y \in \{0, 1, \ldots, m-1\}$ je

$$\text{Prob}\{i \in I \mid f_i(x) = y\} = \frac{1}{m}$$

tak dostáváme, že prvky v k-tici jsou nezávislé. Tedy můžeme použít následující postup.

Na začátku vybereme náhodně s rovnoměrným rozdělením a na sobě nezávisle k funkcí $f_1, f_2, \ldots, f_k$ z nezávislého k-univerzálního systému. Pro reprezentaci použijeme následující algoritmy.

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

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

a do tohoto řetězce přidáme x.

Operace **MEMBER**(x) prohledává řetězce

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

dokud nenalezne x. Pokud neuspěje, pak x není v reprezentované množině.
Pokud je množina vstupů rovnoměrně rozdělena a nezávislá pak můžeme použit předchozí výsledků a dostaneme, že délka všech řetězců je menší než $\frac{\ln \ln n}{\ln k} + O(1)$ s pravděpodobností větší než $1 - o(\frac{1}{n})$, kde n je velikost reprezentované množiny.

Popíšeme použitelnou konstrukci nezávislého 2-univerzálního systému. Nechť p je prvočíslo $q > 1$ je přirozené číslo a univerzum je $U = \{1, \ldots, p^q - 1\}$. Pak existuje bijekce ϕ mezi prvky $u \in U$ a q-ticema čísel $(u_0, u_1, \ldots, u_{q-1})$ z množiny $\{0, 1, \ldots, p-1\}$ taková, že

$$u = \sum_{i=0}^{q-1} u_i p^i \Leftrightarrow \phi(u) = (u_0, u_1, \ldots, u_{q-1}).$$

Pro $j \in \{0, 1, \ldots, q-1\}$ nechť $\phi_j : U \to \{0, 1, \ldots, p-1\}$ je zobrazení takové, že platí $\phi(u) = (\phi_0(u), \phi_1(u), \ldots, \phi_{q-1}(u))$. Pro $a_0, a_1, \ldots, a_{q-1} \in \{0, 1, \ldots, p-1\}$ definujme zobrazení $h_{a_0 a_1, \ldots, a_{q-1}}$ z U do množiny $\{0, 1, \ldots, p-1\}$ takto

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

Položme

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

Tedy velikost $\mathcal{H}$ je p^q. Pak platí

Věta. *$\mathcal{H}$ je nezávisle 2-univerzální systém.*

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

$$a_0, a_1, \ldots a_{q-2}, b_0, b_1, \ldots, b_{q-2} \in \{0, 1, \ldots, p-1\},$$

pak rovnice

$$\sum_{i=0}^{q-2} \phi_i(u)a_i + x\phi_{q-1}(u) \equiv c \bmod p$$
$$\sum_{i=0}^{q-2} \phi_i(u)b_i + y\phi_{q-1}(u) \equiv d \bmod p$$

má právě jedno řešení, protože celá čísla modulo prvočíslo p tvoří těleso. Tedy pro u existuje právě $p^{2(q-1)}$ funkcí z $f_0, f_1 \in \mathcal{H}$ (funkce z různými indexy považujeme za různé) takových, že $f_0(u) = c$ a $f_1(u) = d$. Protože $\phi(u) \neq (0, 0, \ldots, 0)$, existuje i, že $\phi_i(u) \neq 0$ a tedy stačí zaměnit i a $q-1$, takže $\mathcal{H}$ je nezávislé 2-univerzální, protože $\frac{p^{2q}}{p^2} = p^{2(q-1)}$. □

Tedy předchozí výsledek lze použít pro $k = 2$, když tabulka bude mít velikost p pro nějaké prvočíslo p a univerzum bude mít velikost $p^q - 1$ pro nějaké přirozené číslo $q > 0$. Výsledek je možné zobecnit z $k = 2$ na libovolné přirozené číslo $k > 1$.

Srovnejme tento výsledek s výsledkem ze zimního semestru. Tam jsme ukázali, že když funkce $h : U \to \{0, 1, \ldots, m-1\}$ splňuje $|h^{-1}(y) \in \{\lceil \frac{|I|}{m} \rceil, \lfloor \frac{|I|}{m} \rfloor\}$ pro každé $y \in \{0, 1, \ldots, m-1\}$ a když vstupy jsou náhodné a rovnoměrně rozložené, pak očekávaná délka nejdelšího řetězce je $O(\frac{\log n}{\log \log n})$. Tady jsme si ukázali, že když vybereme náhodně s rovnoměrným rozdělením k funkcí z nezávislého k-univerzálního systému a vstupy budou náhodné s rovnoměrným rozdělení, pak očekávaná délka nejdelšího řetězce je $O(\frac{\ln \ln n}{\ln k} + O(1))$. Tedy za použití nezávislého d-univerzálního systému získáváme podstatně lepší výsledek.

Kukaččí hašování..

Kukaččí hašování je varianta perfektního hašování, kde aktualizační operace využívají výsledků a idejí z předchozího paragrafu. Důvodem je, že perfektní hašování zaručuje rychlou realizaci operace **MEMBER** (vyžaduje čas $O(1)$ i v nejhorším případě). Za toto však platíme obtížnou realizací operace **INSERT**. V minulém semestru jsme se setkali s dynamickou verzí perfektního hašování, kde operace **INSERT** a **DELETE** vyžadovaly konstantní očekávaný amortizovaný čas. Aktualizační operace byly založeny na obecných dynamizačních technikách, které dělí reprezentovanou množinu na menší části, viz paragraf Dynamizace. Aktualizační operace pro kukaččí hašování jsou založené na jiné ideji, ale mají podobné charakteristiky složitosti jako dynamizační techniky pro perfektní hašování.

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 $h_1(x)$ 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 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 $\{0, 1, \ldots, m-1\}$ je

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

Když $U = \{0, 1, \ldots, N-1\}$, kde N je prvočíslo a $m = N$, pak polynomy stupně k 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 založené na tomto výsledku.

Popíšeme operaci **INSERT**(x) za předpokladu, že množina 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 na tento řádek vložíme prvek y_0 a skončíme. V opačném případě je $h_1(y_0)$-tý řádek tabulky T_1 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 opakujeme střídavě pro tabulky T_0 a T_1.

Důvodem informace, že x nelze vložit je fakt, 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 $h_0(y_k) = h_0(y_{2i+1})$ pro nějaké $2i+1 < k$, když k je sudé, pak $h_0(y_k) = h_0(y_{2i})$ pro nějaké $2i < k$.

To znamená, že jsme nalezli posloupnost prvku, která se zacyklila a tato operaci by jí jen posunovala v tomto cyklu. To znamená, že v této reprezentaci nelze x vložit do tabulky T_0. V tom případě vrátíme tabulky do původního stavu a stejný postup provedeme s tabulkou T_1 (pokusíme se vložit prvek x do tabulky T_1. Na začátku této procedury předpokládáme, že není označen žádný řádek. Tuto akci realizuje následují podprocedura, 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 máme začít (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
vlož 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 řadek 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 stanovena 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). Nejprve 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**$(0, h_0(x))$. Když úspěšně přerovná prvky, tak končíme. V opačném případě 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 nevedlo **then**
  Vlož$(1, k_1)$
  if podprocedura **Vlož** hlásí, že vnoření se nevedlo **then**
    Výstup: prvek x nelze do této struktury vložit
  endif
endif

Když procedura **INSERT** hlásí, že nelze prvek x vložit do této struktury, 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 pro operaci **INSERT**(y) pro $y \in S \cup \{x\}$.

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 a 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) málo prvků, tak se tabulky zdvojnásobí nebo se zmenší na poloviční.

Autoři ukázali, že když H je $(1, q)$-univerzální systém, pak operace **INSERT** a **DELETE** mají očekávaný amortizovaný čas konstantní.

Tuto metodu navrhli a analyzovali R. Pagh a F. F. Rodler v publikaci z roku 2004.