Takto jsou zobrazeny poznámky cvičících k tomu, co a jak je tu napsáno, skutečný text je mimo oranžové rámečky.

Počítač, na kterém byly provedeny testy: Procesor Intel(R) Core(TM) i7-5557U CPU @ 3.10GHz, RAM 16GB.

Operační systém: Debian 8

Překlad: clang++ verze 3.8, optimalizace -O2.

Konfigurace počítače je důležitá zejména pro měření času. Důležitými parametry jsou: verze procesoru a velikost RAM. Model procesoru můžete na linuxu zjistit příkazem `cat /proc/cpuinfo`. Tyto parametry uvádíme proto, aby všechny testy byly opakovatelné.

1 Test 1 – naivní vs standardní

1.1 Standardní implementace

Je povoleno kouknout se do zdrojového kódu nebo výstupu generátoru a zjistit, co měříme.

První graf (Obrázek 1) zobrazuje průměrný počet kroků na jednu operaci pro různé typy testů a standardní implementaci vektoru. V sekvenčním testu jsme postupně vložili n prvků a následně jsme je zase všechny odebrali. Ve zbývajících dvou testech jsme nejprve vložili n (respektive nejbližší nižší mocninu dvojky) prvků a poté jsme udělali $10n$ náhodných operací v případě náhodného testu nebo jsme $10n$-krát provedli posloupnost dvou vložení a dvou odebrání prvků v případě extrémního testu. Z přednášky víme, že amortizovaná cena jedné operace je konstantní.

Odhadneme asymptotickou složitost operace.

Z grafu naměřených hodnot můžeme vidět, že u všech typů a všech provedených měření můžeme průměrný počet kroků na jednu operaci shora odhadnout konstantou, což souhlasí s tvrzením z přednášky.

Pozor, že děláme odhady toho, co jsme naměřili. V žádném případě z naměřených dat nedostáváme důkaz, že se datová struktura bude chovat dle našich představ i pro vyšší, než námi naměřené hodnoty. Dokonce ani pro měřené velikosti nedostáváme horní odhad chovaní datové struktury. Takové tvrzení by bylo třeba zdůvodnit (důkazem z přednášky, intuicí...).

Odhadneme multiplikativní konstantu.

Z grafu můžeme shora odhadnout multiplikativní konstantu okolo 1.5 kroku na operaci, která vychází pro sekvenční test. Povšimněme si, že vektor dosahoval mnohem lepších výsledků pro náhodný a extrémní test, kde bylo průměrně na operaci potřeba udělat 0.2 kroky. Konstanta nám vychází menší než 1, protože do kroků nezapočítáváme vložení prvků, ale pouze každé překopírování prvku, při změně velikosti vektoru.

Všimneme si, co je v grafech za zvláštnosti a pokusíme se je zdůvodnit.

U všech tří křivek si můžeme všimnout, že přibližně na mocninách dvojky dojde k prudkému nárustu počtu kroků, který se potom postupně snižuje k další mocnině dvojky. Nejvíce se toto chování projevuje u sekvenčního testu. Domnívám se, že je to způsobeno tím, že pro tuto velikost dojde k naalokování většího pole a překopírování prvků mezi novým a starým polem. Při vkládání více prvků do již alokované volné paměti není potřeba alokovat více místa a proto se průměrný počet kroků na operaci snižuje. U náhodného a extrémního testu nejsou tyto skoky tak výrazné a domnívám se, že jsou způsobeny úvodní inicializační fází a poté se udělá spousta operací, které nevyvolají realokaci.

1.2 Naivní implementace

Přidáváme i graf s logaritmickou škálou, protože nárust počtu kroků v extrémním případě způsobí, že nevíme jak vypadají ostatní křivky.

Na dalších třech grafech (Obrázek 2, 3 a 4) můžeme pozorovat chování naivní implementace. V prvním grafu můžeme vidět, že na rozdíl od správné implementace průměrný počet kroků na jednu operaci v našem měření roste v extrémním testu lineárně s velikostí držené množiny. Multiplikativní konstanta se pohybuje mezi jednou polovinou a jednou čtvrtinou (viz Obrázek 4).

Myslím si, že k lineárnímu nárustu dochází díky tomu, že struktura je nucena často přealokovávat celé pole. Domnívám se, že k tomu dochází díky vhodně zvolené velikosti držené množiny (vždy držíme množinu velikosti nejbližší menší mocniny dvojky) a posloupnosti příkazů (střídání operace push a pop).

Rozdílné je také chování u náhodného testu, ve kterém se v případě naivní implementace objevují mnohem větší peaky, pokud velikost držené množiny je přibližně mocnina dvojky. Výšku peaků bychom z grafu mohli odhadnout pomocí odmocniny z n s multiplikativní konstantou $\frac{3}{10}$. Tato funkce je znázorněna v grafu (Obrázek 4) šedou křivkou. Toto celkem sedí s teorií náhodných procházek na celých číslech.

Všimnout si jevu je třeba a je třeba to napsat! Zdůvodnění mimo selského rozumu a znalostí z přednášek třeba není, tedy odhad rychlosti růstu jako odmocnina a zmínka o náhodných procházkách není nutná.

Ideální by nejspíš bylo změřit tyto hodnoty i okolo větších mocnin dvojky, ale bylo by velmi časově náročné to udělat pro více než dvě další. Další možností by bylo udělat více testů pro stejnou velikost a udělat průměr (případně zkusit, jestli je správná distribuce hodnot dle teorie náhodných procházek).

Návrh dalších experimentů testujících naše hypotézy také není špatný.

Dále si můžeme všimnout, že pro velmi malé velikosti množiny standardní implementace v sekvenčním i extrémním testu o trošku roste, ale na druhou stranu v náhodném testu trošku klesá. Myslím, že je to dané jednak tím, že jsme úvodní velikost vektoru zvolili osm, to se projeví zejména v testech, při kterých se velikost vektoru jen zvětšuje a pak jen zmenšuje. Dalším vlivem může být to, že pro malé vektory je větší pravděpodobnost několika operací push a několika operací pop tak, že se velikost vektoru změní (pokud stojíme v nule a podle hodu mincí jdeme o krok vpravo nebo vlevo, pak v 100 krocích častěji dojdeme do 10 než do 80).

Znovu zdůvodnění nad rámec znalostí z přednášky a základní pochopení chování datové struktury není nutné.

Dále si můžeme všimnout drobného nárůstu naivní implementace viditelného na Obrázku 2. Nevím, čím je toto způsobené.

Není třeba zdůvodnit naprosto všechno, jen to, co se přímo týká teorie nebo je jednoduše vymyslitelné.

1.3 Shrnutí

V sekvečních testech vykazovaly obě struktury podobné vlastnosti, ale v ostatních testech se chovala standardní struktura lépe. Nejvýraznější byl rozdíl v extrémním testu, ve kterém standardní implementace dosahovala konstantní složitosti okolo 0.06, zatímco naivní implementace roste lineárně v závislosti na velikosti držené množiny. V náhodném testu dosahovaly obě struktury shodných výsledků, vyjma případů, kdy velikost držené mocniny byla okolo mocniny 2. V těchto případech si správná struktura udržovala konstantní průměr počtu kroků na jednu operaci, zatímco v případě naivní struktury průměr rostl s odmocninou velikosti množiny.

2 Test 2 – push s různými c

Z grafů znovu pozorujeme kolísání na mocninách nafukovacího parametru c (předtím bylo pozorováno na mocninách $c = 2$) a pak pozvolné klesání. Vysvětlení je obdobné, na mocnině c znovu alokujeme a pak přidáváme další prvky do prázdného místa, ale nic za ně nealokujeme, pouze jejich počtem dělíme.

Dále si můžeme všimnout, že zdola jsou křivky odhadnuté zhruba $\frac{1}{c-1}$ a shora $1 + \frac{1}{c-1}$, což odpovídá právě alokaci na mocninách c a větě z přednášky.

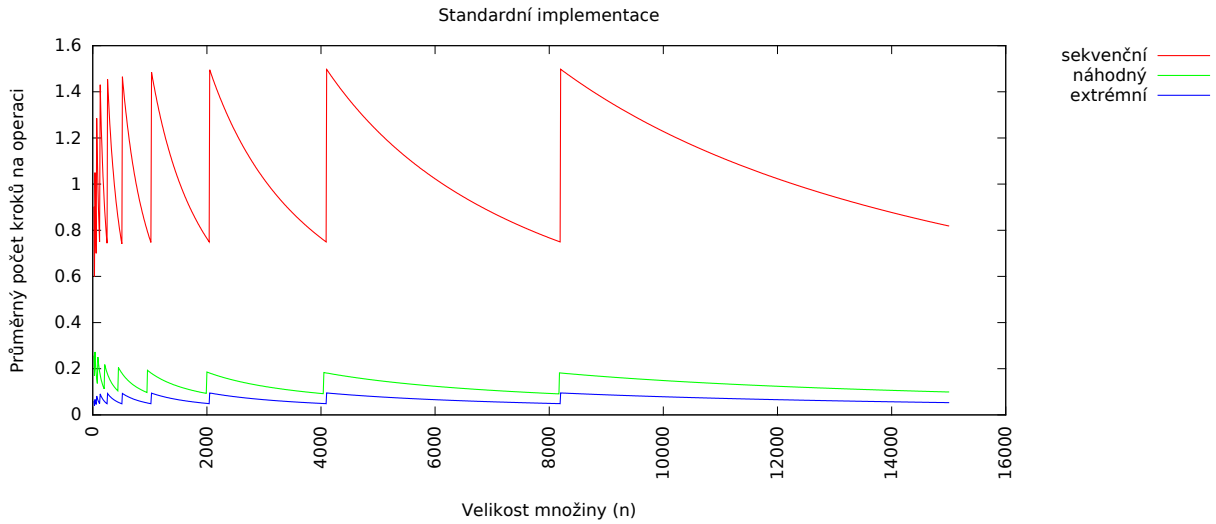
Všimneme si také, že pro $c = 1.1$, pro které by mělo platit předchozí pozorujeme drobný nárůst. Ze začátku grafu je nárůst výrazný, pak se zmenšuje. Z grafů nedokážu poznat, jestli se konverguje k nějaké hodnotě nebo ne. Na přednášce jsme dokázali, že křivka konverguje ke konstantě.

Na grafech je hezky vidět, že čísla, která jsou svými mocninami mají některé skoky společné (například 2,4,8 nebo 3,9). To odpovídá tomu, že mocnina 8 je také mocninou 2.

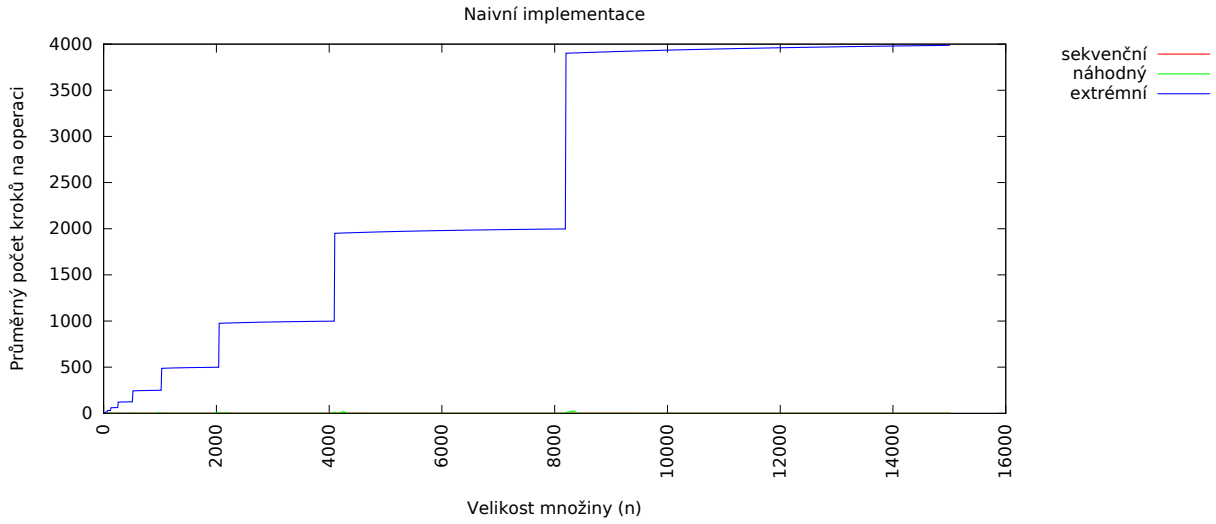
Pozorování byla provedena na základě Obrázku 7 a Obrázku 8. Logaritmické měřítko uvádíme jen pro ukázku.

3 Grafy

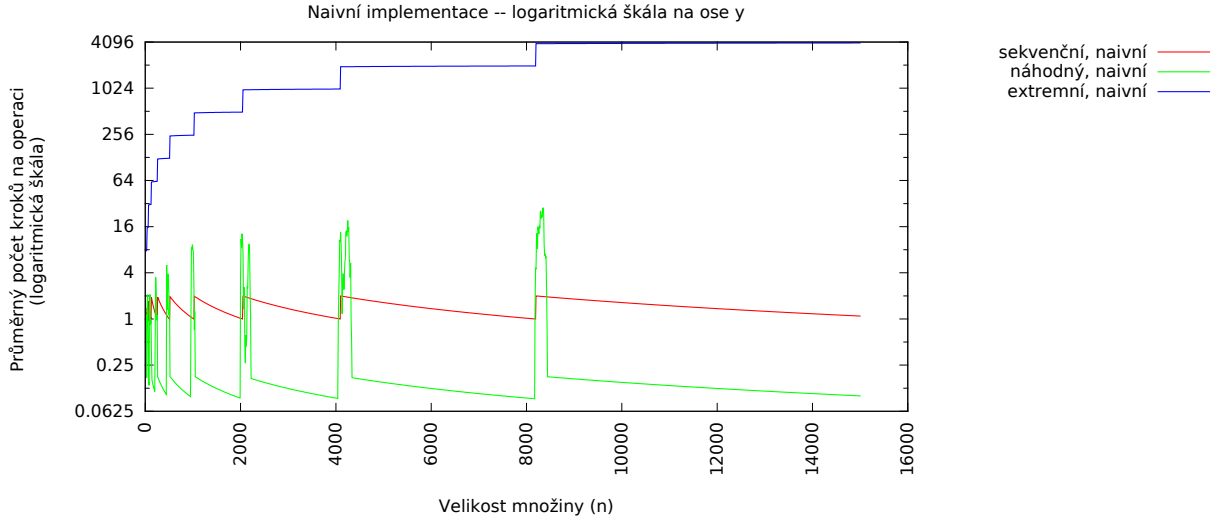
3.1 Grafy Test 1



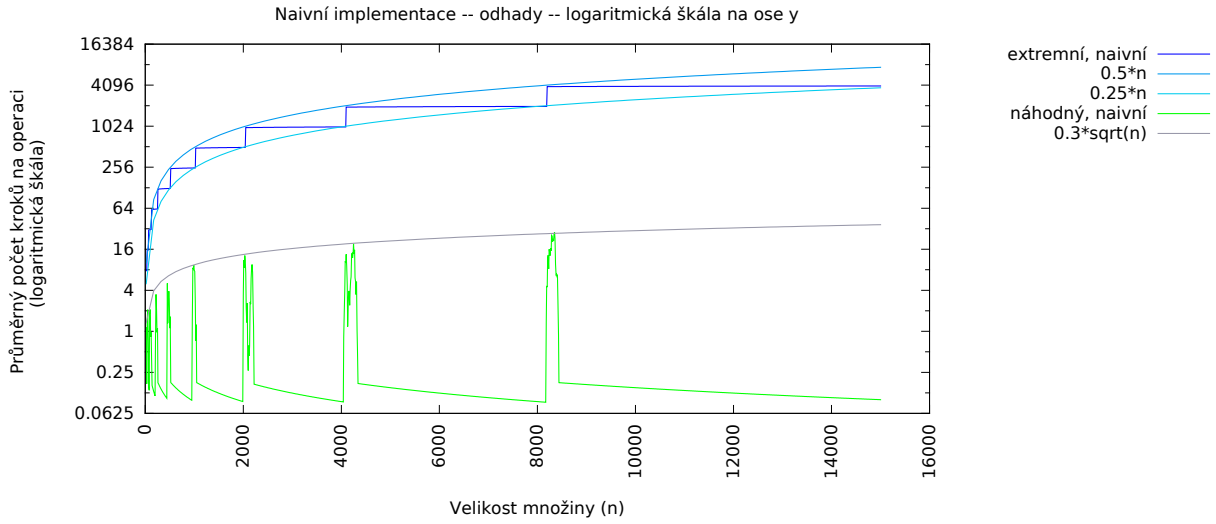
Obrázek 1: Správná implementace.



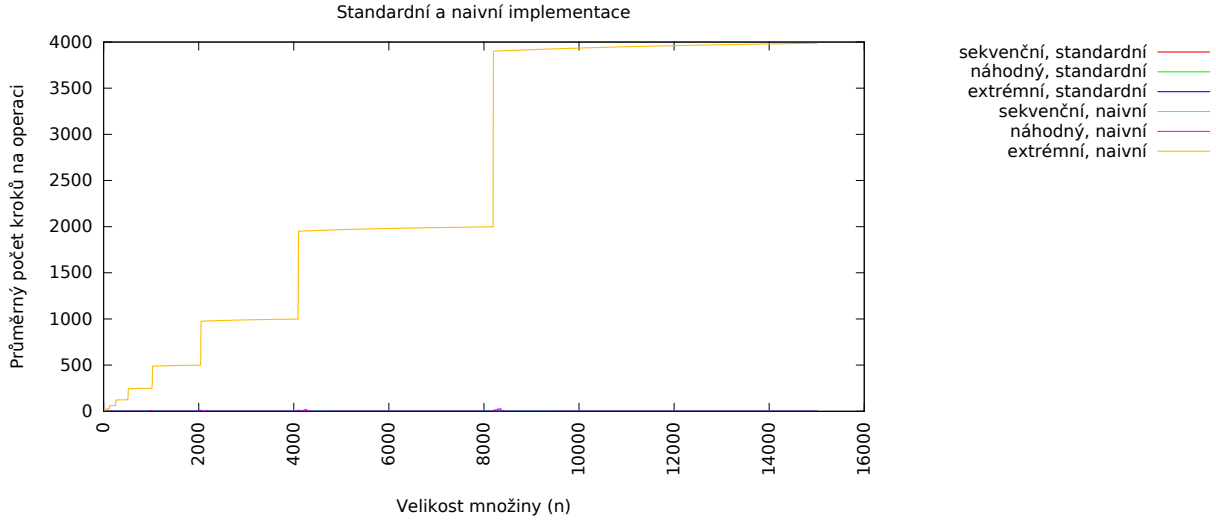
Obrázek 2: Naivní vs správná implementace.



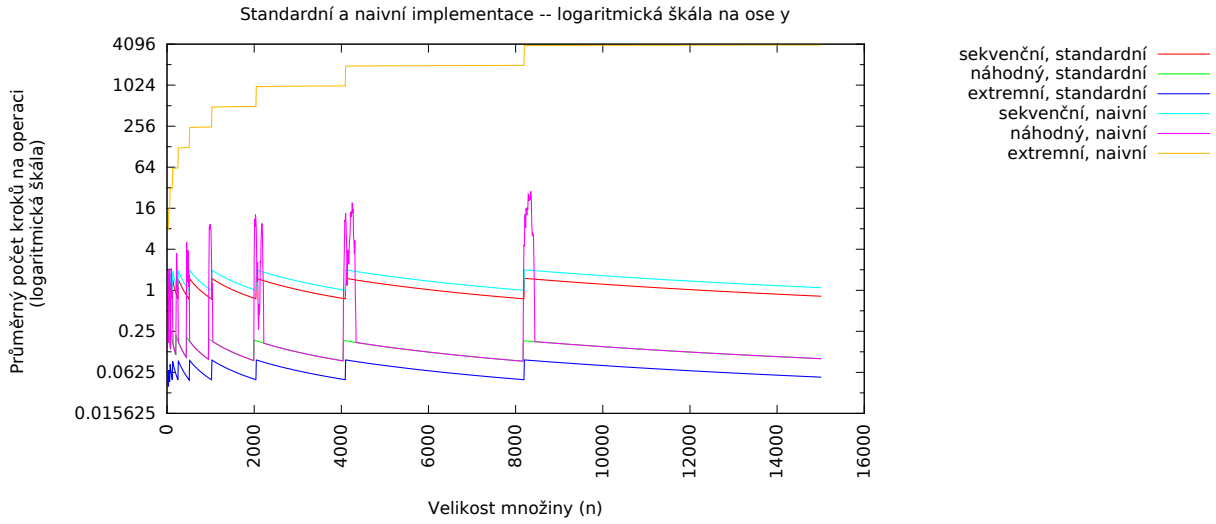
Obrázek 3: Naivní vs správná implementace s logaritmickou škálou.



Obrázek 4: Naivní vs správná implementace s logaritmickou škálou.

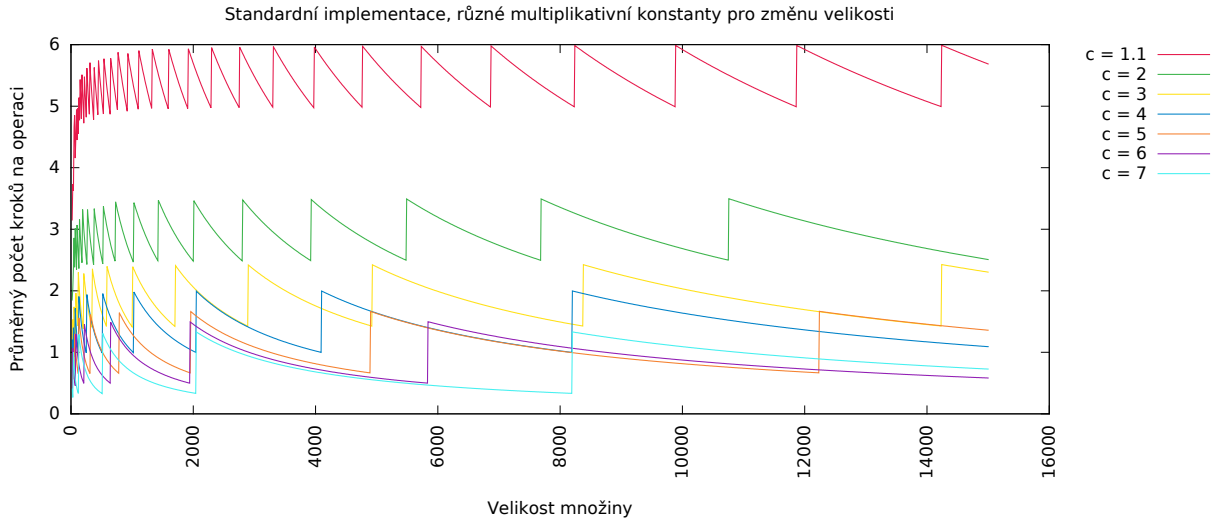


Obrázek 5: Naivní vs správná implementace.

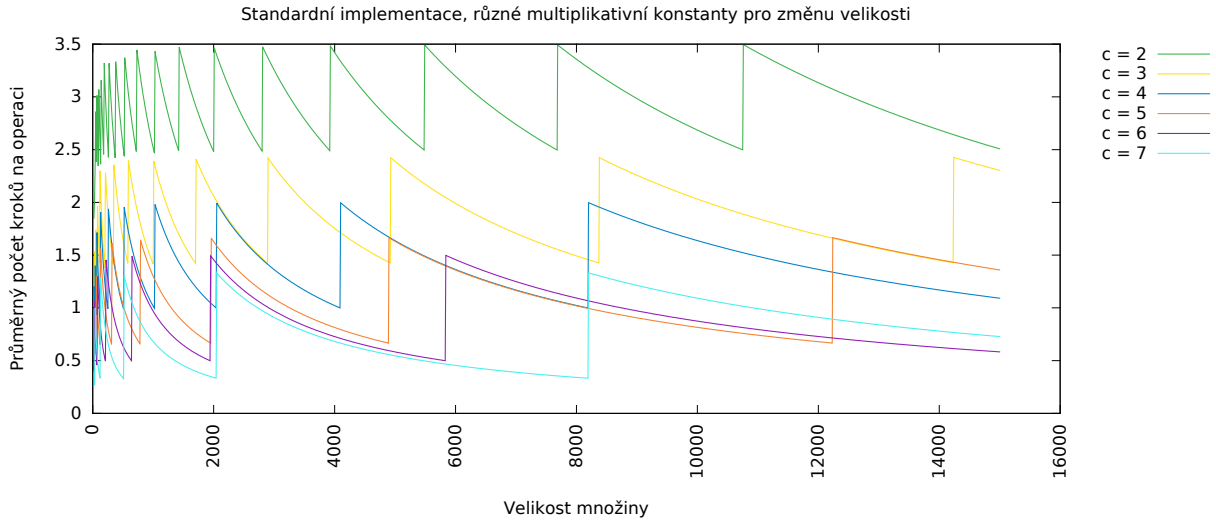


Obrázek 6: Naivní vs správná implementace.

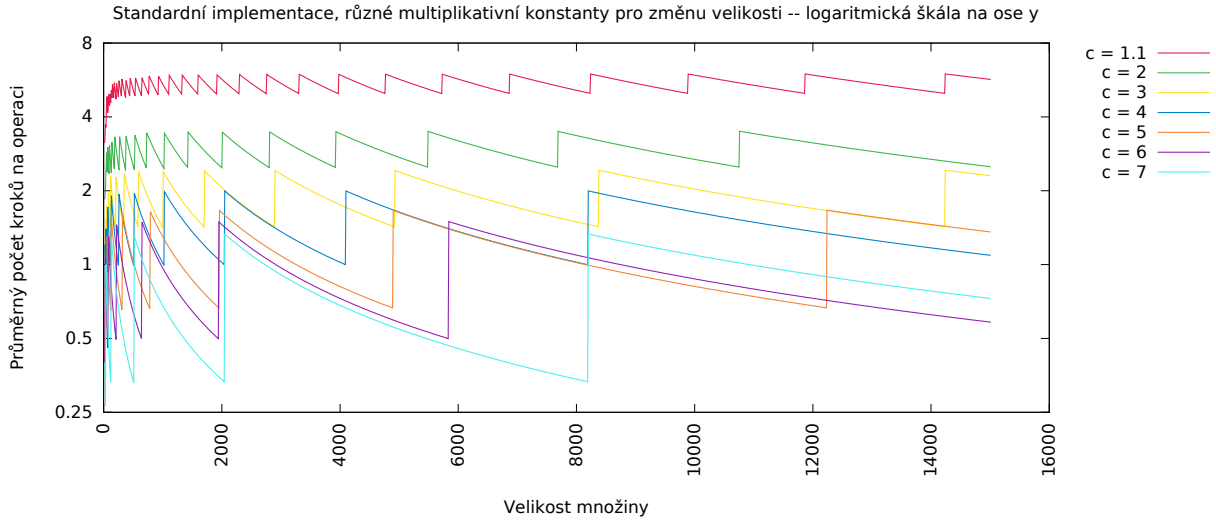
3.2 Grafy 2



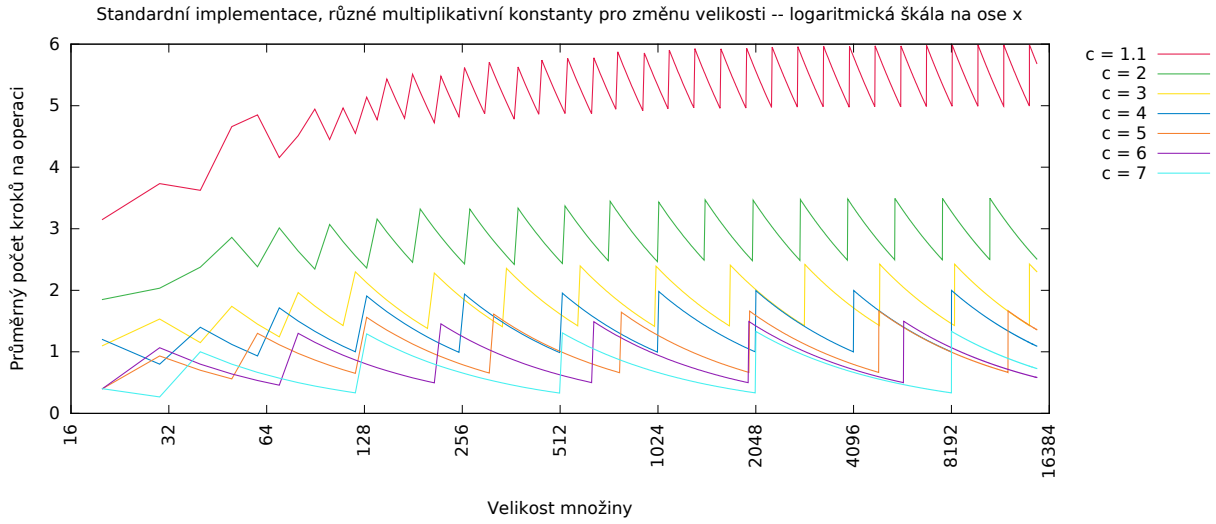
Obrázek 7: Různé zvětšování alokovaného prostoru.



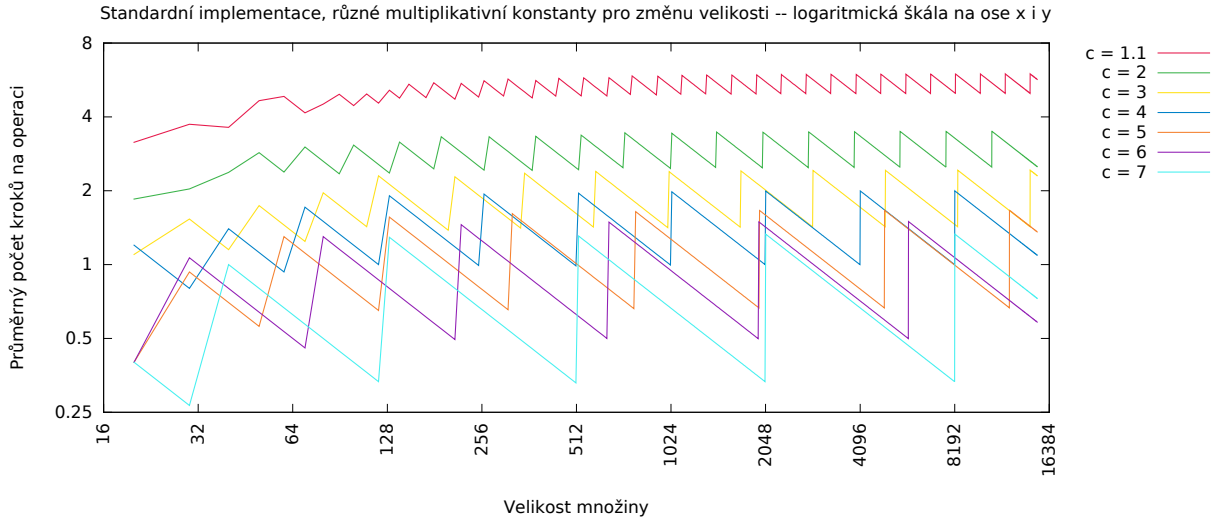
Obrázek 8: Různé zvětšování alokovaného prostoru.



Obrázek 9: Různé zvětšování alokovaného prostoru.



Obrázek 10: Různé zvětšování alokovaného prostoru.



Obrázek 11: Různé zvětšování alokovaného prostoru.