INFORMAČNÍ BULLETIN

České statistické společnosti

Ročník 35, číslo 3, září 2024

Obsah

Informační bulletin České statistické společnosti vychází čtyřikrát do roka v českém vydání. Příležitostně i mimořádné české a anglické číslo. Vydavatelem je Česká statistická společnost, IČ 00550795, adresa společnosti je Na padesátém 81, 100 82 Praha 10. Evidenční číslo registrace vedené Ministerstvem kultury ČR dle zákona č. 46/2000 Sb. je E 21214. Časopis je sázen v programu TEX, ve formátu LuaHBTEX s písmy balíku CSfonts.

The Information Bulletin of the Czech Statistical Society is published quarterly.
The contributions in the journal are published in English, Czech and Slovak languages.

DOI: 10.5300/IB, http://dx.doi.org/10.5300/IB
ISSN 1210–8022 (Print), ISSN 1804–8617 (Online)

Toto číslo bylo vytištěno s laskavou podporou Českého statistického úřadu.

ÚVODNÍK
FOREWORD

Redakce

Kdo si ~~ne~~hraje, nezlobí!

Vážené čtenářky, vážení čtenáři našeho bulletinku,

připravili jsme pro Vás číslo s velkou měrou zaměřené na rekreační matematiku. Mohlo by to být pro Vás užitečné do výuky či jako inspirace na jiný typ příkladů.

V prvním bloku je článek na generování pseudonáhodných čísel po cifrách, což se hodí pro případy, kdy je počet cifer obrovské číslo. To je případ například Rubikovy kostky pro delší hrany. Ale ne zas tak velké, aby to nezvládl výpočetní stroj.

K tomu tématicky spadá recenze na knihu *Permutation Puzzles*.

Druhý blok je zaměřen na logickou hru sudoku. Použitelný postup pro sudoku s překryvy (angl. Multi-Sudoku Puzzle) přes program Sugen je představen v prvním článku.

Nezávislé ověření na jedinečné řešení takové obrovské struktury je proveden v programu Picat, to je představitel řešitelů SAT, to je zmíněno v druhém článku.

Třetí článek je zaměřen na vykreslení takových typů sudoku přes knihovnu Raylib, ba co víc, je ukázána instalace knihovny i pro výpočetní prostředí R.

Tento největší blok doplňují recenze na knihy *Taking Sudoku Seriously* a *Geometric Magic Squares* jako nový typ ve světě magických čtverců (angl. Geomagic). Blok uzavírá rešerše zdrojů se zaměřením na tuto hru.

Poslední blok patří novinkám ve světě TEXu, ať už se jedná o nové balíčky či větší aktualizace balíčků existujících.

Jako perlička je zmíněna rešerše zdrojů kolem vyřešeného dlouholetého geometrického problému (v březnu a květnu 2023): nalezeného jednoho dílku, který neperiodicky pokryje nekonečnou plochu.

Inspirativní čtení přeje,
Pavel Stříž

Slavkov u Brna, na sv. Valentýna, 14. 2. 2024

GENEROVÁNÍ PSEUDONÁHODNÉHO ČÍSLA PO CIFRÁCH
PSEUDORANDOM NUMBER GENERATION BY DIGITS

Pavel Stříž

E-mail: pavel@striz.cz

Abstrakt: Článek představuje jednoduchý algoritmus na generování pseudonáhodného čísla po cifrách (brány zleva doprava). Interval bere v rozsahu $[0;N-1]$, kde N se bere jako jistý počet možností, nejčastěji kombinací, které jsou indexovány od jedničky. Autor zmiňuje svůj postřeh, že je potřeba jen cca prvních deset cifer, zbytek generovaných cifer lze rozhodit na další výpočetní stroje, protože se volí čísla v rozsahu [0;9] bez dodatečné podmínky. Princip algoritmu by byl stejný i pro binární čísla. Zdrojové kódy jsou v článku přidány, pro Python3 a pro R. V závěru příspěvku autor zmiňuje nastavení svého oblíbeného editoru SciTE, tedy aby zdrojový kód v R byl barevně zvýrazněn a bylo možné jej z editoru spouštět.

Klíčová slova: PRNG, R, Python, C, SciTE.

Abstract: The article introduces an simple algorithm which generates a pseudorandom number by digits (from left to right). The interval of the number is $[0;N-1]$, where N is a number of certain situations, most often combinations (indexed from 1). The author states that we only need about first 10 digits, then we can distribute problem to different computers because we generate digits from interval [0;9] without any condition. The idea of the algorithm would be the same for binary numbers. The source code is included, in Python3 and in R. In the conclusion for user's convenience, the author mentiones his gained experience in setting up the SciTE editor to activate syntax highlighting and turn on the option of running the R code within the editor.

Keywords: PRNG, R, Python, C, SciTE.

1. O problému

Při řešení Rubikovy kostky: výpočtu kombinací, výběru jedné z nich a zobrazení Rubikovy kostky, a obráceně, po zobrazení jisté Rubikovy kostky zjistit pořadové číslo kombinace, jsem zjistil, že počet kombinací u větších Rubikových kostek dosahuje obřích čísel. U řešené virtuální kostky (světový rekord, hrana délky $n = 2^{16}$) se dostáváme na počet kombinací nad 16 miliard cifer. To už není na počítání, řekl bych.

U Rubikovy kostky už jen pro délku hrany $n = 2^{11} = 2048$ dostáváme počet možností N o 16257517 cifrách, můžete si se svém oblíbeném programu na výpočty ověřit. V závěru článku na str. 12 představuji řešení pro Python.

$$N = 7! \cdot 3^6 \cdot (24 \cdot 2^{10} \cdot 12!)^{n \bmod 2} \cdot 24!^{\left\lfloor \frac{n-2}{2} \right\rfloor} \cdot \left(\frac{24!}{4!^6} \right)^{\left\lfloor \left(\frac{n-2}{2} \right)^2 \right\rfloor}$$

Dokonce i zamíchat takovou kostku se touto cestou už skoro nedá, musí se na to jít bez výpočtu možností, či zkusit rozhodit skupiny barev, jak toho využívá program RCube. Zamíchání Rubikovy kostky dle sekvence [FBLRUD] není totiž rovnoměrně náhodné.

I tak jsem si říkal, že by bylo zajímavé si vygenerovat pseudonáhodné číslo v intervalu $[0;N-1]$, kde N je počet možností (indexován od jedničky).

2. První úvaha

Asi nejjednodušší cesta je vzít počet cifer takového obrovského čísla. Vygenerovat si sérii cifer 0–9 a na konci srovnat s $N-1$. Je-li vygenerovaná hodnota menší či rovna $N-1$, uznáme ji, v opačném případě generujeme celou sekvenci znovu.

3. Druhá, ošklivá úvaha

Asi nejrychlejší výpočetní podvod je vzít první cifru a ponížit ji o jedničku. Na první cifře generovat v rozsahu [0;dopočtené], všechny ostatní cifry pak už v příjemném rozsahu [0;9]. Například u čísla 6855 dostáváme 0–5 na první cifře a 0–9 na zbylých, tedy dostáváme čísla v rozsahu 0000–5999. Bohužel za tu cenu, že nikdy nezískáme hodnotu v rozsahu 6000–6855. Úvahu, s úsměvem, opusťme jako nepatřičnou.

Pokud bychom měli hovořit o ošklivosti, tak ještě horší, byť na první pohled funkční případ, je užít na každé cifře interval nula až daná cifra, např. u hledání čísla [0;634] volit 0–6 na první cifře, 0–3 na druhé a 0–4 na třetí. Zdánlivě je vše v pořádku, ale to je omyl, např. číslo 552 nelze získat díky druhé cifře.

4. Lepší nápad

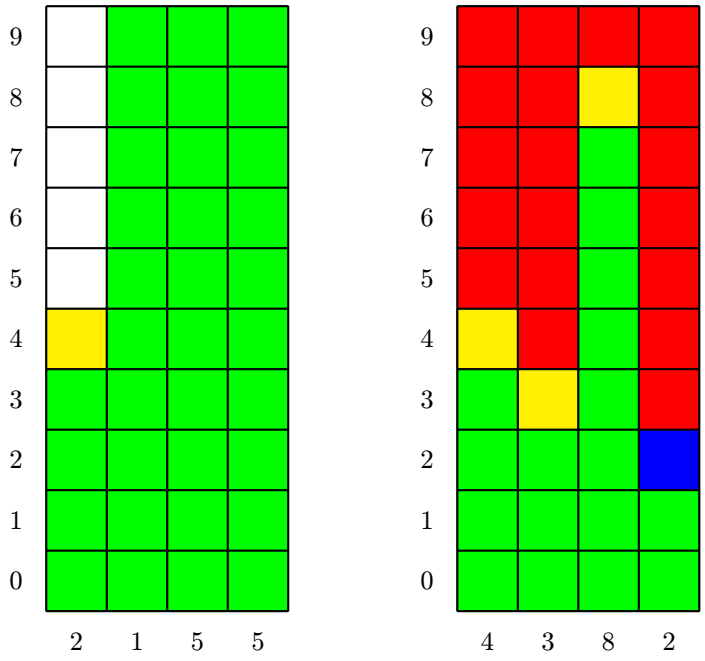
První úvaha je použitelná, ale časově náročná, neb řadu hodnot musíme vyřadit. Jistá myšlenka byla, jak by se tato situace dala zefektivnit. Algoritmus by se dal popsat slovy takto. Vnější cyklus jde přes cifry, zleva doprava (nejvyšší řád mocniny po nejnižší). Vygenerujeme číslo v intervalu [0;9] a srovnáme

s cifrou na zkoumané pozici. Pokud je menší, u všech dalších cifer může být [0;9] a zmíněná podmínka není dále nutná. Pokud je cifra stejná, hodnotu přijmeme, ale podmínka zůstává zachována pro další cifry. A pokud je cifra vyšší, celý pokus rušíme a začneme znovu.

Výjimkou je první cifra, tam s jistotou víme, že generovaná cifra musí být v intervalu jedna až zkoumaná první cifra, nemusíme se tedy držet intervalu [0;9].

Poznámka. Kdybychom místo [0;9] užili interval nula až zkoumaná cifra při neaktivní podmínce, dostali bychom se do podmíněné pravděpodobnosti, to je nad rámec tohoto příspěvku. I tak je to slepá ulička.

Zde je ukázka hodnoty z [0;4282]. U čísla 2155, jsme u první cifry zjistili, že je menší než 4, ostatní mohou získat libovolné hodnoty. Naopak u čísla 4282 samotného jsme s podmínkou stále aktivní i po čtyřech cifrách. Například číslo 4400 bychom vyřadili na druhé cifře, 6000 a vyšší již na první cifře apod.



Například u čtyřciferného čísla (1000–9999) je nejhorší možnost právě 1000. Vyřadíme hodnoty 1001–9999 (cca 90 % hodnot), u čísla 9999 bychom nevyřadili pokus žádný. V závislosti na zvoleném N však můžeme říci, že

v průměru vyřazujeme 45 % hodnot. Poněvadž ale vyřazení pokusu uděláme obvykle dřív než u posledních cifer, stálo za hřích zkusit simulaci, na které cifře ke zrušení podmínky sledování nižší hodnoty dochází. Tím se totiž zajistí, že už k dalšímu vyřazení nemůže dojít.

Cifra	Přij. abs.	Přij. rel.	Přij. abs. kum.	Přij. rel. kum.
1	8800861	0,8800861	8800861	0,8800861
2	1071983	0,1071983	9872844	0,9872844
3	114334	0,0114334	9987178	0,9987178
4	11499	0,0011499	9998677	0,9998677
5	1173	0,0001173	9999850	0,9999850
6	132	0,0000132	9999982	0,9999982
7	14	0,0000014	9999996	0,9999996
8	4	0,0000004	10000000	1,0000000
9	0	0,0000000	10000000	1,0000000
10	0	0,0000000	10000000	1,0000000
$\sum$	10000000	1,0000000	—	—

Ze simulovaných 10 miliónů čísel o deseti cifrách (hledané číslo jen jednou) vidíme, že jsme potřebovali prvních osm cifer. Další cifry byly o výběru z 0–9, což se dá distribuovat na jiné výpočetní stroje. Při 100 miliónech čísel jsem se dostal na dva případy na deváté cifře.

Poněvadž se generování opakuje od začátku, když je generovaná hodnota při aktivní podmínce na cifře vyšší než hledaná, pro badatele by mohla být zajímavá tato tabulka, která zmiňuje počet pokusů než se generování celého čísla podařilo.

Pokusů	Případů	Pokusů	Případů
1	8927265	11	57
2	869600	12	23
3	148173	13	17
4	36748	14	7
5	11385	15	3
6	4017	16	1
7	1604	17	0
8	706	18	0
9	283	19	0
10	111	20	0

Nejlepší statistika pokusů je u čísla se samými devítkami (vždy první úroveň, není kde číslo vyřadit), nejhorší pak s číslem začínajícím jedničkou a zbytek nuly (dochází i na úroveň mnoha desítek pokusů).

Možností tohoto algoritmu je, že nám stačí generování bitů. Ze sekvence bitů si už libovolně velké číslo $N-1$ složíme. Princip metody je však stejný jako u čísel desítkové soustavy.

Pokud tedy generujeme číslo o tisících cifrách a mnohem větší, je nám už jedno, jestli na úvod uřízneme prvních osm, deset či dvacet cifer. Důležité pro nás bylo vědět, kde alespoň přibližně se máme pohybovat. S každou dodatečnou cifrou se řádově posouváme o řád u zrušení podmínky. 10 miliónů čísel u této simulace je osmiciferné číslo, tedy řádově počítáme hranici osm uřízlých cifer plus nějaká rezerva. Je to dané tím, že posun mezi ciframi bez zrušení podmínky zajišťuje jedna hodnota z deseti možných, tedy 10 %.

5. Poznámky k programům

Jednoduchý program byl prvně naprogramován v Pythonu3 a pak za pomoci https://www.javainuse.com/py2r převeden do R a za pomoci webové stránky https://extendsclass.com/python-to-javascript.html převeden do JavaScriptu. Zdrojový kód pro JavaScript zde neuvádím.

```
import random
hodnota=4383 # Indexování od 1 -- N.
cislo=str(hodnota-1) # Indexování od 0 -- N-1.
kolik=40 # Kolik hodnot si přeji vygenerovat.
print("Generuji",kolik,"hodnoty od nuly po",cislo,"...")
delka=len(cislo)

def generuj():
    while True:
        odCisla=0; retezec=""
        doCisla=int(cislo[0]); mensi=False
        poradi=0; znovu=False
        while poradi<delka:
            nahodne=random.randint(odCisla,doCisla)
            doCisla=9
            if nahodne<int(cislo[poradi]):
                mensi=True
            if nahodne<=int(cislo[poradi]) or mensi:
                retezec+=str(nahodne)
            else:
                znovu=True; break
            poradi+=1
        if not znovu:
            nalezene=int(retezec)
            if nalezene<hodnota:
                print(retezec); return()
for _ in range(kolik): generuj()
```

Kdybychom v Pythonu3 chtěli skutečně pracovat s obřími čísly, nikoliv textovým řetězcem hned od začátku, užijeme:

```
import sys
sys.set_int_max_str_digits(0)
```

Při online konverzi jsem zjistil, že Python3 bere `return` i `return()`, výstup se mi u R zacyklil. Konvertítko chybně bralo „ě“, u „í“ zahlásilo chybu.

Pro badatele ve výpočetním prostředí R přikládám zdrojový kód.

```
# Generování PRN po cifrách...
hodnota <- 4383 # Indexování od jedničky, 1 -- N.
cislo <- as.character(hodnota - 1) # Indexování od nuly, 0 -- N-1.
kolik <- 40 # Počet chtěných cifer.
print(paste("Generuji", kolik,"hodnot(y) od nuly po", cislo, "..."))
delka <- nchar(cislo)

generuj <- function(){
  while(TRUE){
    odCisla <- 0; retezec <- ""
    doCisla <- as.numeric(substr(cislo, 1, 1))
    mensi <- FALSE

    poradi <- 0; znovu <- FALSE
    while(poradi < delka){
      nahodne <- sample(odCisla:doCisla, 1)
      doCisla <- 9
      if(nahodne < as.numeric(substr(cislo,poradi+1,poradi+1))){mensi <- TRUE}
      if(nahodne <= as.numeric(substr(cislo,poradi+1,poradi+1)) || mensi){
        retezec <- paste(retezec, nahodne, sep = "")
      } else { znovu <- TRUE; break }
      poradi <- poradi + 1
    }

    if(!znovu){
      nalezene <- as.numeric(retezec)
      if(nalezene < hodnota){
        print(retezec); return()
      }
    }
  }
}
for(invalue in 1:kolik){generuj()}
```

6. Simulujme!

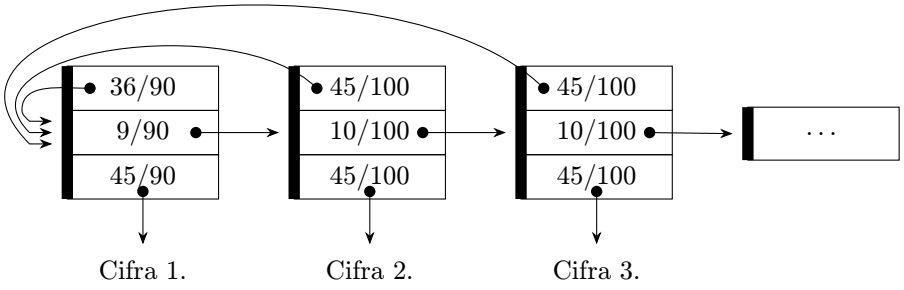
Věřím tomu, že se jedná o zajímavou pravděpodobnostní úlohu výpočtu přesné hodnoty, s jakou pravděpodobností na cifře dojde k vyřazení podmínky; úloha však byla a je nad autorovy síly. Kdyby na to náhodou někdy, někdo došel, pochlubte se.

Zkusím však celou situaci zjednodušit. Na každé úrovni se rozhodujeme mezi třemi stavy. Pokus opakovat (hodnota vyšší než chtěná), posunout se na

další úroveň (hodnota je stejná jako žádaná) a nakonec uzavřít experiment (zapadnutí čísla do úrovně, zbytek je už jen generování cifer 0–9, tedy pro nás nezajímavé). V tabulce je souhrn všech možností, šipka dolů je uložení úrovně, šipka vlevo opakovat pokus a šipka vpravo zvednutí úrovně.

Žádaná \ Generovaná	0	1	2	3	4	5	6	7	8	9
0	→	←	←	←	←	←	←	←	←	←
1	↓	→	←	←	←	←	←	←	←	←
2	↓	↓	→	←	←	←	←	←	←	←
3	↓	↓	↓	→	←	←	←	←	←	←
4	↓	↓	↓	↓	→	←	←	←	←	←
5	↓	↓	↓	↓	↓	→	←	←	←	←
6	↓	↓	↓	↓	↓	↓	→	←	←	←
7	↓	↓	↓	↓	↓	↓	↓	→	←	←
8	↓	↓	↓	↓	↓	↓	↓	↓	→	←
9	↓	↓	↓	↓	↓	↓	↓	↓	↓	→

Tím dostáváme 45 situací (opakovat), 10 situací (posunutí o úroveň) a dalších 45 situací (uzavření čísla na dané úrovni), zkráceně 45-10-45. Jedinou výjimkou je první úroveň, kdy pro n-ciferné číslo předpokládáme, že nezačíná nulou, tím odpadá první řádek a dostáváme 36-9-45.



Pro Céčkaře něco na závěr. Tímto pokusem jsem se dostal na jeden případ na desáté cifře, a asi nejpřesněji, jak jsem to jen dovedl.

```
// gcc -o 45-10-45 45-10-45.c && ./45-10-45
#include <stdio.h>
#include <stdlib.h>
int main() {
long maxH=1000000000;
char uroven, nahodna;
long urovne[16]; for (char k=0; k<16; k++) {urovne[k]=0;}
srandom(0);
for (long hodnot=0; hodnot<maxH; hodnot++) {
    uroven=0;
```

```
    while (1) {
        uroven++;
        if (uroven>1) {
            nahodna=random()%100;
            if (nahodna<45) {urovne[uroven]++; break;}
            if (nahodna>54) {uroven=0;}
        } else {
            nahodna=random()%90; // zásah
            if (nahodna<45) {urovne[uroven]++; break;}
            if (nahodna>53) {uroven=0;} // zásah
        } // if
    } // while
} // for
for (char k=1; k<16; k++) {printf("%d %ld %0.016f\n",
    k, urovne[k], (double)urovne[k]/maxH);}
return 0;
}
```

Tiše sedím, sázím tento článek, a říkám si, že bych nemusel čísla takto posílat po jednom, že je to neefektivní, ale rovnou zaslat celý blok. A poposouvat čísla do hloubky také v blocích, kam jen to jde, vždyť struktura je známá. A z hodnot zasílaných zpět začít znovu, dokud se nám nepodaří vše rozhodit. Zde je na závěr kód pro Pythonisty.

```
import math
zaklad=10000000000000000000; varka=zaklad
urovne=[0]*21; zbytky=[0]*21
def hloubeji(kolik,uroven): # Rekurze
    global varka
    kolik=kolik+zbytky[uroven]
    zpet=math.floor(kolik*45/100)
    poslidal=math.floor(kolik*10/100)
    posliuroven=math.floor(kolik*45/100)
    urovne[uroven]+=posliuroven
    zbytky[uroven]=kolik-zpet-poslidal-posliuroven
    varka=varka+zpet
    if poslidal>0: hloubeji(poslidal,uroven+1)
while True:
    uloz=math.floor(varka*45/90+varka*36/90*45/(9+45))
    dal=math.floor(varka*9/90+varka*36/90*9/(9+45))
    varka=varka-dal-uloz # nová várka
    urovne[1]+=uloz
    if dal>0: # Popošli blok čísel...
        hloubeji(dal,2)
    else:
        # Uzavření výpočtů...
        zbytky[1]=varka; varka=0; break
soucet=0
for uroven in range(1,len(urovne)):
    urovne[uroven]+=zbytky[uroven] # Závěrečné přičtení zbytků k úrovni.
```

```
soucet+=urovne[uroven]
print(uroven, urovne[uroven],
    format(urovne[uroven]/zaklad,"0.020f"),
    soucet, format(soucet/zaklad,"0.020f"))
```

Pole `zbytky` je pomůcka, jak si dávat pozor na zaokrouhlování čísel. Na úplný závěr se přičtou k dané úrovni, ale jedná se už jen o jednotky. Myslím, že struktura rozhození do úrovní přes všechny případy je už čitelná. Matematik by řekl, že se jedná o (ne)konečné řetězové zlomky.

Cifra	Přijatých absolutně	Přijatých relativně	Přij. abs. kumulovaně	Přij. rel. kumulovaně
1	90909090909091044	0,0909090909090909606	9090909090909091044	0,9090909090909090
2	818181818181818044	0,0818181818181818038	9909090909090909088	0,9909090909090909
3	81818181818181812	0,0081818181818181807	9990909090909090900	0,9990909090909091
4	8181818181818182	0,0008181818181818181	9999090909090909082	0,9999090909090908
5	818181818181819	0,0000818181818181819	9999909090909090901	0,9999909090909090
6	81818181818181	0,0000081818181818181	9999990909090909082	0,9999990909090908
7	8181818181819	0,0000008181818181819	9999999090909090901	0,9999999090909090
8	818181818183	0,0000000818181818183	9999999090909090084	0,9999999090909090
9	81818181819	0,0000000081818181819	9999999990909090903	0,9999999990909090
10	8181818181	0,0000000008181818181	9999999999090909084	0,9999999999090909
11	818181820	0,0000000000818181820	9999999999090909904	0,9999999999090909
12	81818182	0,0000000000081818182	9999999999990909086	0,9999999999990909
13	8181819	0,0000000000008181819	9999999999999090905	0,9999999999999090
14	818182	0,0000000000000818182	9999999999999909087	0,9999999999999908
15	81819	0,0000000000000081819	9999999999999990906	0,9999999999999991
16	8182	0,0000000000000008182	9999999999999999088	0,9999999999999998
17	819	0,0000000000000000819	9999999999999999907	1,0000000000000000
18	83	0,0000000000000000083	9999999999999999990	1,0000000000000000
19	9	0,0000000000000000009	9999999999999999999	1,0000000000000000
20	1	0,0000000000000000001	10000000000000000000	1,0000000000000000
∑	10000000000000000000	1,0000000000000000000	—	—

7. Implementace

Kdybychom něco takového skutečně potřebovali a i při všech těchto znalostech, nejlepší je se riziku vyhnout. Riziko v tomto případě znamená, že se některý blok čísel bude brát v ohledu vícekrát. Buď tedy ať výpočetní stroj generuje všechny cifry, nebo po ukončení podmínky, ať si jiným strojům řekne *kolik* cifer ještě chybí. Než to s rizikem dělat tak, že si na začátku řekneme, že budeme generovat např. 10, 20 či 30 cifer na jednom stroji a zbytek na jiných.

Zde je výpočet počtu cifer ze začátku článku ze str. 4. Funguje to v Pythonu verze 3.10.12. Převod na celá čísla, `int()`, v posledním činiteli je tam záměrně, abychom po operaci dělení nepracovali s desetinným číslem, to má svá omezení.

```
import math
import sys; sys.set_int_max_str_digits(0)
n=2**11 # Základní Rubikova kostka by byla n=3.
x=math.factorial(7) * 3**6 * \
  (24 * 2**10 * math.factorial(12))**(n%2) * \
  math.factorial(24)**(math.floor((n-2)/2)) * \
  (int(math.factorial(24)/(math.factorial(4)**6)))** \
  (math.floor(((n-2)/2)**2))
print("Délka hrany =", n, ", počet cifer =", len(str(x)))
```

Poznámka. Tato práce s velkými čísly (angl. Arbitrary Number Precision) je stále otevřený problém v C a C++. Uživatel musí sáhnout do jiných zdrojů mimo oficiální knihovny či si to musí naprogramovat sám. V ukázce v Pythonu či v Javě (knihovna `BigNumber`) to už takový problém není.

8. Poznámky k editoru SciTE místo Závěru

Je to k nevíře, ale do řešení této úlohy jsem svůj oblíbený editor (SciTE) na editaci a spuštění R kódu nepoužil. Instalujeme si jej případně přes (Ubuntu):

```
sudo apt install scite
```

spuštění přes `scite`, nebo přes:

```
flatpak install flathub org.scintilla.SciTE
```

spuštění přes `flatpak run org.scintilla.SciTE`.

Abychom dostali formátovaný soubor napsaný v R, musíme na konci souboru `/usr/share/scite/SciTEGlobal.properties` vyhodit R z filtru (zapsán malým písmenem er). Předdefinovaný v menu v nabídce programů není.

Pro spuštění (klávesa F5) jsem si v `~/.SciTEUser.properties` přidal:

```
command.go.$(file.patterns.r)=Rscript $(FileNameExt)
```

Nemohu to na 100 % potvrdit ani vyvrátit, ale v nové verzi SciTE lze přes Ctrl a plus či mínus zvětšovat a zmenšovat písmo, to si v dřívějších verzích nevybavuji. Po bližším zkoumání zjišťuji, že to tím není. Funguje to s plus a mínus na numerické klávesnici, nikoliv s těmi ze základní sady písmen. To jsem si vylepšil v `~/.SciTEUser.properties` přidáním (2333 je kód pro ZoomIn, 2334 pro ZoomOut, to jsem vyčetl ze souboru `CommandValues.html` z dokumentace tohoto programu):

```
user.shortcuts=Ctrl++|2333|Ctrl+-|2334|
```

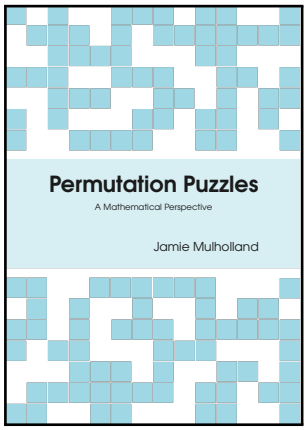
Vítejte v eRkovém a výpočetním světě!

RECENZE KNIHY: PERMUTATION PUZZLES – A MATHEMATICAL PERSPECTIVE
BOOK REVIEW: PERMUTATION PUZZLES – A MATHEMATICAL PERSPECTIVE

Pavel Stříž

E-mail: pavel@striz.cz

Jamie Mulholland: *Permutation Puzzles. A Mathematical Perspective,* 1. vydání, vlastním nákladem, 2021, 337 stran. Vedle titulní strany je zmíněno 12 parit Rubikovy kostky, str. 251, a 12 rotačních symetrií jehlanu, str. 271.



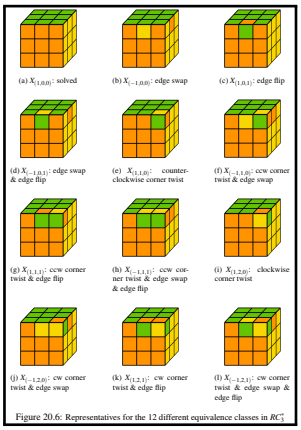


Figure 20.6: Representatives for the 12 different equivalence classes in RC_3^3

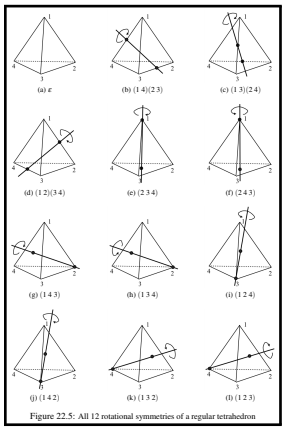
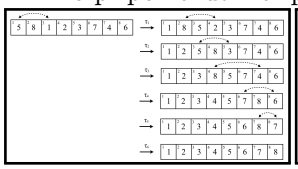


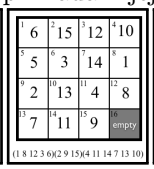
Figure 22.5: All 12 rotational symmetries of a regular tetrahedron

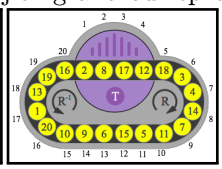
Tuto knihu, shrnující poznatky z přednášek kurzu *Math 302 – Mathematics of Permutations Puzzles*, http://www.sfu.ca/∼jtmulhol/math302/, jsem potkal již v roce 2016 jako verzi předběžnou k této knize. Vůči knize jsem změny nezaznamenal, jedná se hlavně o grafickou úpravu.

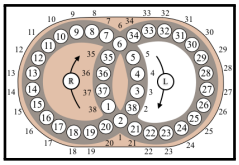
Kniha čtenáře, resp. studenty, uvádí do teorie grup. Výhodou je, že nevyužívá jen Rubikovu kostku, ale další čtyři hry: Swap, 15-Puzzle, Oval Track Puzzle a Hungarian Rings (číslovanou a čtyřbarevnou verzi). S tím, že Rubikova kostka těžké partie vždy završuje. Speciální skupinu tvoří v knize hra Light's Out, zmiňuje ukázku práce se SageMath při užití lineární algebry.

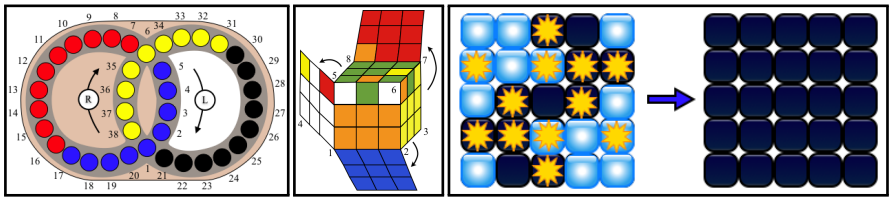
Pro připomenutí her přikládám jejich grafickou reprezentaci:











Kniha od kapitoly 13. Komutátory, str. 161, doporučuje užívat pomůcky, hry ve fyzické či alespoň virtuální podobě. Kombinace vedoucí k přesunu jen určitých částí Rubikovy kostky je asi opravdu dobré si zkusit v rukách.

Další výhodou knihy je, že představuje SageMath (http://www.sagemath.org/) napsaný v Pythonu a dílčí partie jsou v knize řešeny.

Každá kapitola obsahuje řadu úloh a cvičení. Většinou bez řešení. Formálně je kniha rozdělena na šest částí a přílohy: základy (kap. 1 a 2), permutace (kap. 3–9), teorie grup (kap. 10–18), Rubikova kostka (kap. 19–21), symetrie a počty rozdílných typů (kap. 22 a 23) a Light's Out.

Vedle Seznamu literatury a Rejstříku kniha v přílohách obsahuje Úvod do výpočetního prostředí SageMath a Základní vlastnosti celých čísel.

Potěšila mě kapitola o komutátorech, tedy hledání postupů, jak vyřešit Rubikovu kostku. To hodně připomíná hráčské metody řešení kostky.

Poněvadž jsem si úlohy k Rubikově kostce programoval (ranking-unranking problem), potěšila mě analýza, proč je při náhodném rozebrání a složení Rubikovy kostky řešitelná jen každá dvanáctá, v úvodu recenze prostřední obrázek s typy parit. V kapitolách 22 a 23 (The Orbit-Stabilizer a Burnside's Theorem) mě potěšilo znázornění všech 12 rotačních symetrií u jehlanu (v úvodu obrázek vpravo).

Mezi zdroji, pravděpodobně i inspirace ke knize, najdeme J. O. Kiltinen: *Oval Track and Other Permutation Puzzles: And Just Enough Group Theory to Solve Them.* New York: The Mathematical Association of America, 2003, 142 pp. ISBN 0-8838-5725-1.

Za typografii se jedná o čistou práci, není co vytknout: vysázené v TEXu, odkazy klikatelné, obrázky nakresleny ve vektorové formě v barvě, podobně jako zdrojové kódy pro SageMath. Nevyzkoušel jsem všechny, ale ty, které jsem vyzkoušel, bez problémů jely. Užita verze SageMath 9.0 z 1. 1. 2020, v pozadí běžící Python verze 3.8.10.

Autor knihu podpořil svými webovými stránkami, http://www.sfu.ca/~jtmulhol/permutationpuzzles, a doporučuje k nahlédnutí i J. Scherphuis: *Jaap's Puzzle Page*, viz http://www.jaapsch.net/puzzles/, kde si lze řadu her, nejen ty zmíněné v knize, virtuálně zahrát.

SUDOKU S PŘEKRYVY: AHOJ, SVĚTE!

MULTI-SUDOKU: HELLO, WORLD!

Pavel Stříž

E-mail: pavel@striz.cz

Abstrakt: Autor stručně představuje proces generování sudoku s překryvy. V pozadí používá program Sugen, který byl pro tyto účely upraven na úrovni jazyka C. Na základním příkladu je proces představen krok za krokem. V závěru autor zmiňuje omezení tohoto algoritmu. V principu můžeme říci, že jakékoliv sudoku s překryvy, které obsahuje cyklus, tímto způsobem nelze generovat, je potřeba jiný přístup, a to přes rekurzivní funkci.

Klíčová slova: C, Sugen, sudoku.

Abstract: The article briefly introduces a process of generating a multi-sudoku. Behind the scene, the author uses Sugen program which was modified at a C level for this specific task. A "Hello, World!" multi-sudoku is presented step-by-step. In the conclusion, the author mentiones its limits, esp. which multi-sudoku cannot be generated by this algorithm. It can be stated that any multi-sudoku with cycle would be a problem, this type of multi-sudoku needs a different approach, a recursion function.

Keywords: C, Sugen, sudoku.

1. Nápad

Řeším kombinatorické úlohy, především ranking-unranking problem, tedy zjistit počet možností, vybrat si jednu z nich, zrekonstruovat kombinatorickou situaci, a naopak, z jisté kombinatorické situace spočítat její pořadové číslo. To mě přivedlo i ke klasickému sudoku, kde jsem řešení nezahlédl, ale první experimenty ukazují, že to bude náročné, ale možné. O tom snad jindy.

Při rešerši kolem sudoku jsem narazil na verze s překryvy (anglicky overlapping sudoku, multi-sudoku). Pravděpodobně nejznámější je Samurai sudoku. Ale existuje jich celá plejáda.

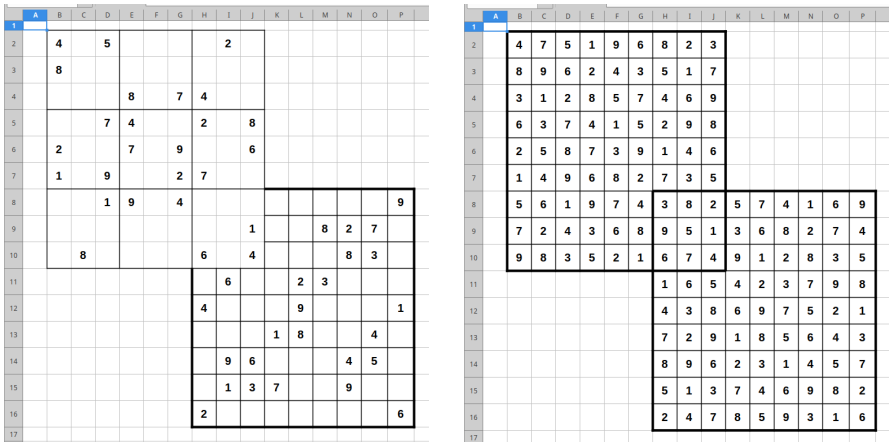
2. Program sugen

Chtěl jsem si něco takového zkusit naprogramovat. Při rešerši jsem narazil na program Sugen (zkráceno ze sudoku generator) od Daniela Beera, https://dlbeer.co.nz/articles/sudoku.html.

Ten umí vygenerovat mustr, zadání sudoku ze zadaného mustru i získat obtížnější variantu řešeného sudoku. Neumí však generovat více sudoku s překryvy. Poněvadž se učím C, tak jsem začal zdrojový kód zkoumat. Zkusit si vygenerovat sudoku s překryvy od nuly mě tehdy ani nenapadlo.

3. Hello, World!

Našim úkolem je připravit obdobu tohoto sudoku s překryvy. Překrývá se celý blok, to lze považovat za standard.



4. Tři injekce od pana doktora (či sestřičky)

Jakmile jsem princip programu pochopil, použil jsem načtení externích souborů před zavoláním klíčové funkce. Připadal jsem si jako správný hacker.

1. injekce. Číslo sudoku slouží jako počáteční hodnota PRNG, funkce `srandom`. Pro případ pozdního nového generování jen jednoho sudoku. Nebylo třeba. Pomocný soubor obsahuje jednu hodnotu.

2. injekce. Pro vytvoření řešení, funkce `choose_grid` jsem zasahoval do pole `grid`. Pomocný soubor má dva sloupce, číslo pole (0–80) a hodnota (1–9). Vzniká soubor s řešením.

3. injekce. Pro vytvoření zadání, funkce `harden_puzzle` jsem zasahoval do pole `puzzle`. Pomocný soubor má dva sloupce, číslo pole (0–80) a jeho hodnotu (0 – musí zůstat prázdné, či 1–9). Plus se načte soubor s řešením. Výstupem je soubor se zadáním.

Poznámka. Dá se tak řešit i chtěná obtížnost, nebo program umožňuje si zavolat parametr `-t` z příkazového řádku. Toho jsem využil.

5. Dílčí kroky

Překryv je pro první sudoku 9. blok (pravý dolní roh), pro druhé sudoku blok 1. (levý horní roh). To když má člověk pořád na mysli, sledovat proces generování není pak tak náročné, hlavně u složitějších struktur.

Schématicky bychom generování mohli znázornit takto.

Upravený sugen: 1. sudoku, řešení:

```
_________                       475196823
_________                       896243517
_________                       312857469
_________                       637415298
_________ --> bez injekce --> 258739146
_________                       149682735
_________                       561974382
_________                       724368951
_________                       983521674
```

Upravený sugen: 1. sudoku, zadání:

```
475196823                       4_5____2_
896243517                       8________
312857469                       ___8_74__
637415298                       __74__2_8
258739146 --> bez injekce --> 2__7_9__6
149682735                       1_9__27__
561974382                       __19_4___
724368951                       ________1
983521674                       _8____6_4
```

Upravený sugen: 2. sudoku, řešení:

```
_________      382______       382574169
_________      951______       951368274
_________      674______       674912835
_________      _________      165423798
_________ ---> _________ ---> 438697521
_________      _________      729185643
_________      _________      896231457
_________      _________      513746982
_________      _________      247859316
```

Upravený sugen: 2. sudoku, zadání:

```
382574169       000______       ________9
951368274       001______       __1__827_
674912835       604______       6_4___83_
165423798       _________       _6__23___
438697521 --->  _________ --->  4___9___1
729185643       _________       ___18__4_
896231457       _________       _96___45_
513746982       _________       _137__9__
247859316       _________       2_______6
```

Z příkazového řádku jsem si postupně dvakrát volal:

```
$ ./sugen gen-grid >reseni.txt
$ ./sugen harden -t 100 <reseni.txt >zadani.txt
```

Pro první sudoku byly dva pomocné soubory prázdné, pro druhé sudoku s naznačenými hodnotami u injekcí.

Poslední úkol je už čistě typografický, výstupní soubory si vhodně vysázet.

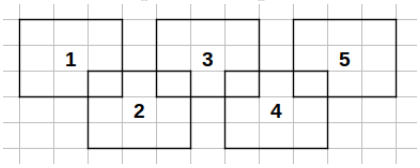
6. Kdy to nelze použít

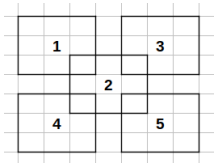
Je zde omezení. Jakmile bychom měli komplikovanější strukturu překryvů (sudoku vztahy jakoby ob jedno sudoku), dříve či později se dostaneme do konfliktu, kdy musíme sudoku generovat znovu, či samozřejmě lépe, využít rekurze. O tom více na přednášce.

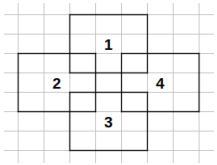
Pokud však najdeme pořadí sudoku, abychom jedno negenerovali ze vztahů více nezávislých sudoku, tato metoda nám dostačuje. Termínem z teorie grafů, do každého sudoku může jít maximálně jedna šipka, ze sudoku ven libovolný počet. Šipky určují pořadí generování.

Pro ilustraci dva náčrtky. Na levém obrázku můžeme generovat sudoku např. v těchto pořadích: 1–2–3–4–5, 5–4–3–2–1 nebo třeba 3–2–1–4–5. Do problémů se dostáváme např. při 1–2–3–5–4 či 5–1–2–3–4, ale také při pokusu generovat sudoku jakoby po řádcích, tedy 1–3–5–2–4.

Na středním obrázku můžeme využít: 1–2–3–4–5 či 2–1–3–4–5, ale dostaneme se do problémů při 1–3–2–4–5 či 1–3–4–5–2.



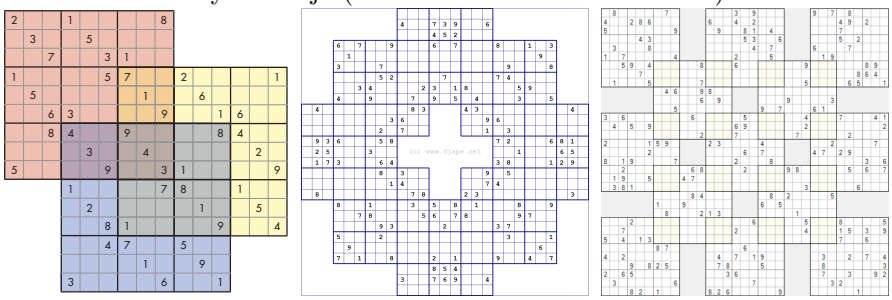




Tato pomůcka sledu generování sudoku nám nepomůže u multi-sudoku, které tvoří uzavřený cyklus, kdy každé sudoku má víc nezávislých překryvů, viz pravý obrázek.

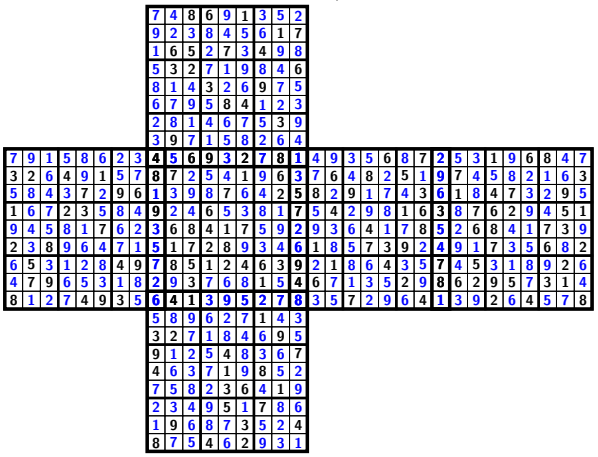
Na takovou situaci se už musí jít přes rekurzivní funkci. Či mnou nedoporučený způsob vyřazování nevhodného sudoku a opakování. V takovém případě se ale může stát, že řešení ani nalézt nelze a musí se jít v procesu generování o celé sudoku nazpět. Chce to jiný přístup, o tom víc na konferenci.

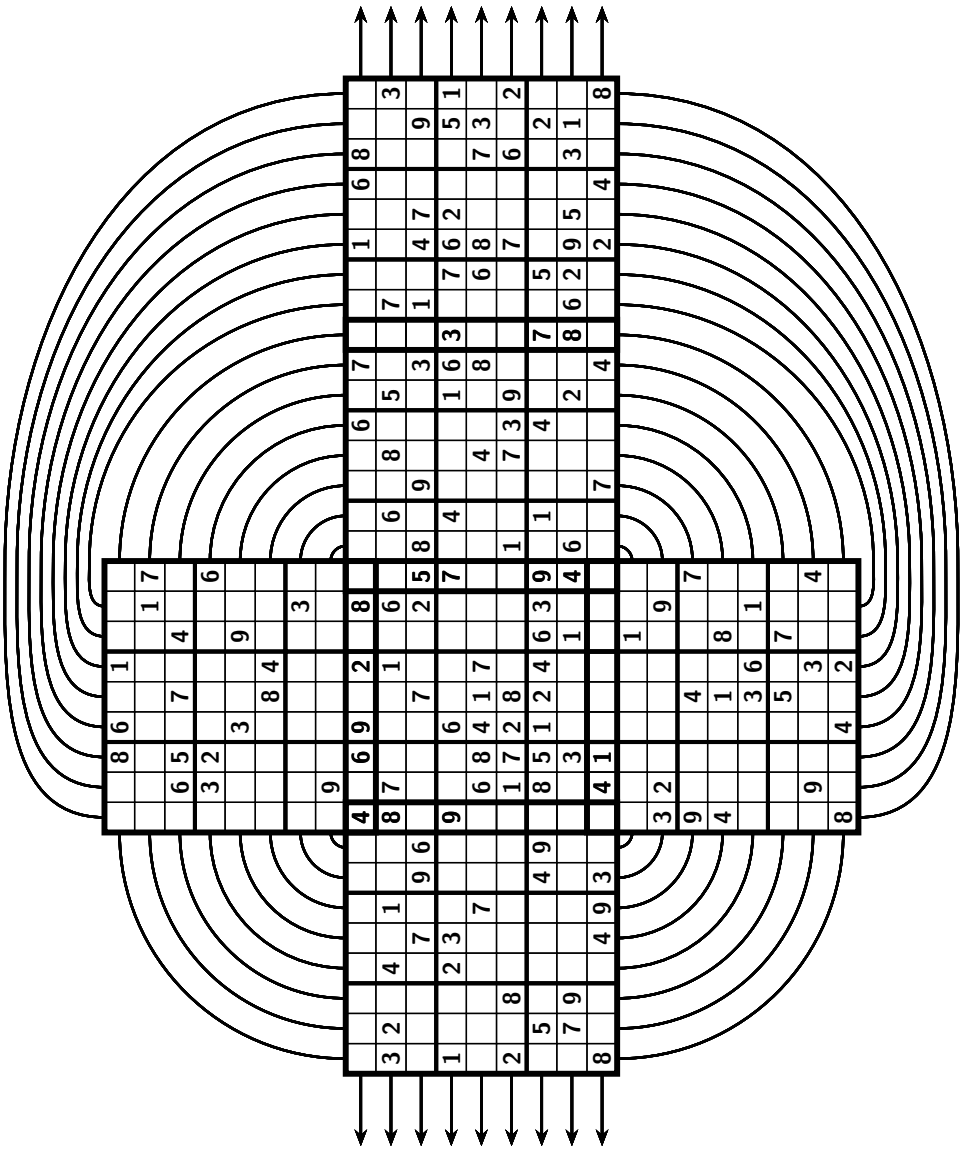
Tento typ jsem neprogramoval, neb jsem to zatím nepotřeboval, ale kdyby na to došlo, zde je tzv. Venn sudoku (*Taking Sudoku Seriously*, str. 122), které je ideálním typem na testy. Plus další dva rozsáhlejší cyklické typy přebrané z internetu z různých zdrojů (Sudoku Gattai a Sudoku Sumo).



7. Vzorek z vlastní zahrádky

Na této straně je řešení, na další straně zadání 3D sudoku (povrch krychle, tedy 6 sudoku), rozkresleného na ploše se vztahy. Vztahy jsou komplikovanější (hrany sdílí dvě sudoku, rohy dokonce tři), ale princip vzniku je podobný.





ÚVOD DO PROGRAMU PICAT
INTRODUCTION TO THE PICAT PROGRAM

Pavel Stříž

E-mail: pavel@striz.cz

Abstrakt: Článek stručně představuje proces ověření sudoku s překryvy, kde je nutné mít jen jedno řešení, v programu Picat. Jako ukázka slouží pět sudoku složených do podoby olympijských kruhů s překryvy velikosti jednoho bloku.

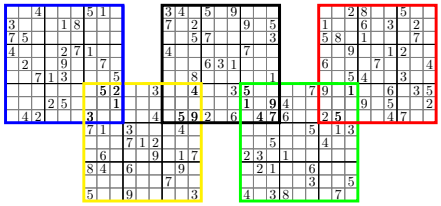
Klíčová slova: Sudoku, Picat, řešitel SAT.

Abstract: The article briefly describes a process of multi-sudoku verification, where one unique solution is necessary, in program Picat. An exemplary multi-sudoku are olympic rings formed by 5 sudokus with single block overlaps.

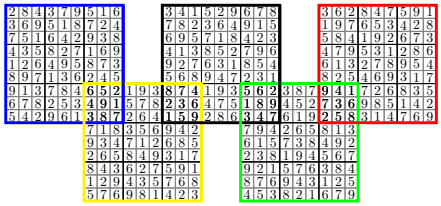
Keywords: Sudoku, Picat, SAT solver.

1. Bádání nad sudoku s překryvy

U svých experimentů nad multi-sudoku (viz druhý příspěvek v tomto sborníku) jsem se dostal do bodu, kdy jsem již byl spokojený s výstupy. A to do takové míry, že jsem připravil 5 sudoku ve tvaru olympijských kruhů a nabídl to Informačnímu bulletinu České statistické společnosti jako PF 2024 k otištění, viz IB 4/2023.



Zadání multi-sudoku.



Řešení multi-sudoku.

Ověřil jsem jedinečné řešení dílčích sudoku, přes program `sugen`, ale později také přes program `sudoku` (GitHub: KyleGough). Vynechal jsem takové, které vyžadovaly víc kol řešení, byť to asi nebylo třeba. Zkušený řešitel sudoku lítá z jednoho sudoku do druhého, vlastně takový teoretický počet nutných kol vůbec neřeší.

2. Ověření celku

Prozradím, že tento krok není potřeba, protože, když jsou dílčí sudoku s jedním řešením, musí být s jedním řešením i celek. To mě tehdy nenapadlo. Navíc šlo o mé první publikování takového ražení, chtěl jsem mít jistotu, že tam není víc řešení.

Dříve jsem zkoumal program Picat, kde mezi mnoha stovkami ukázek bylo i nalezení řešení klasického sudoku. Říkal jsem si, že by měl jít ověřit i mnohem větší projekt. Zajímalo mě též, jak dlouho to bude trvat.

3. Program Picat

Program spadá do kategorie SAT řešitelů, není ve standardním linuxovém repozitáři, ale dá se nainstalovat po stažení z http://picat-lang.org.

Syntaxe jazyka je nestandardní, ale dá se v ní rychle zorientovat. Nezvyk je, že dílčí úseky jsou oddělovány čárkami, závěr příkazu je ukončen tečkou, podobně jako složené věty v běžném jazyce.

Během generování řešení a zadání multi-sudoku se mi generuje i celý program pro Picat (užívám Lua a Python3), který se následně spustí.

4. Ukázka kódu

Generovaný soubor je ve finále obrovský, využívám `[...]` kvůli zkrácení pro účely vydání.

Soubor si pojmenujme `kruhy.pi` a část proměnných by vypadala takto. Proměnných je celkem 405, pět sudoku po 81 proměnných. Pořadí jsem měl dané zleva doprava: A pro modrou, B pro žlutou, C pro černou, D pro zelenou a E pro červenou. Zkušené oko vidí, že by se na překryvech daly proměnné ušetřit, ale kvůli čitelnosti kódu jsem to nedělal.

Tato část nám definuje proměnné, první písmeno je sudoku, první cifra řádek, druhá cifra pak sloupec daného sudoku. Proměnné omezujeme na cifry 1–9.

```
import sat.
main=>
Vars=[
A11,A12,A13,A14,A15,A16,A17,A18,A19,
[...]
E91,E92,E93,E94,E95,E96,E97,E98,E99
],
Vars :: 1..9,
```

Druhý obrovský logický blok je definování pravidel sudoku, tedy že se cifry 1–9 mohou opakovat jen jednou v řádku, sloupci a bloku jednotlivých sudoku.

```
all_different([A11,A12,A13,A14,A15,A16,A17,A18,A19]),
[...]
all_different([A11,A21,A31,A41,A51,A61,A71,A81,A91]),
[...]
all_different([A11,A12,A13,A21,A22,A23,A31,A32,A33]),
[...]
```

Další obrovský blok je definování překryvů. Tedy že jistá buňka z jednoho sudoku musí být stejná jako buňka jiného sudoku. Takto by to vypadalo pro naše olympijské kruhy.

```
A77#=B11,A78#=B12,A79#=B13,A87#=B21,A88#=B22,
  A89#=B23,A97#=B31,A98#=B32,A99#=B33,
B17#=C71,B18#=C72,B19#=C73,B27#=C81,B28#=C82,
  B29#=C83,B37#=C91,B38#=C92,B39#=C93,
C77#=D11,C78#=D12,C79#=D13,C87#=D21,C88#=D22,
  C89#=D23,C97#=D31,C98#=D32,C99#=D33,
D17#=E71,D18#=E72,D19#=E73,D27#=E81,D28#=E82,
  D29#=E83,D37#=E91,D38#=E92,D39#=E93,
```

Poslední velký blok je zapsání vlastního zadání multi-sudoku.

```
A13#=4, A17#=5, A18#=1, A21#=3, A25#=1, A26#=8,
A31#=7, A32#=5, A41#=4, A45#=2, A46#=7, A47#=1,
A52#=2, A55#=9, A58#=7, A63#=7, A64#=1, A65#=3,
A69#=5, A78#=5, A79#=2, A84#=2, A85#=5, A89#=1,
A92#=4, A93#=2, A97#=3,
[...]
```

Závěr programu obvykle tvoří výpis všech možností, nebo zjištění počtu řešení. Pro úplnost tohoto článku nechávám vypsat obojí.

```
L=findall(Vars,solve(Vars)),
writeln(L),
D=count_all(solve(Vars)),
writeln(D).
```

5. Spuštění programu

Nezbývá než vygenerovaný soubor `kruhy.pi` spustit. Z příkazového řádku by to bylo:

```
$ ./picat kruhy
```

Nebo interaktivně přímo z programu Picat:

```
$ ./picat
Picat> load("kruhy")
[...]
Picat> main
[[2,8,4,3,7,9,5,1,6,3,6,9,5,1,8,7,2,4,7,5,1,6,4,2,9,3,8,4,3,5,8,
2,7,1,6,9,1,2,6,4,9,5,8,7,3,8,9,7,1,3,6,2,4,5,9,1,3,7,8,4,6,5,2,
6,7,8,2,5,3,4,9,1,5,4,2,9,6,1,3,8,7,6,5,2,1,9,3,8,7,4,4,9,1,5,7,
8,2,3,6,3,8,7,2,6,4,1,5,9,7,1,8,3,5,6,9,4,2,9,3,4,7,1,2,6,8,5,2,
6,5,8,4,9,3,1,7,8,4,3,6,2,7,5,9,1,1,2,9,4,3,5,7,6,8,5,7,6,9,8,1,
4,2,3,3,4,1,5,2,9,6,7,8,7,8,2,3,6,4,9,1,5,6,9,5,7,1,8,4,2,3,4,1,
3,8,5,2,7,9,6,9,2,7,6,3,1,8,5,4,5,6,8,9,4,7,2,3,1,8,7,4,1,9,3,5,
6,2,2,3,6,4,7,5,1,8,9,1,5,9,2,8,6,3,4,7,5,6,2,3,8,7,9,4,1,1,8,9,
4,5,2,7,3,6,3,4,7,6,1,9,2,5,8,7,9,4,2,6,5,8,1,3,6,1,5,7,3,8,4,9,
2,2,3,8,1,9,4,5,6,7,9,2,1,5,7,6,3,8,4,8,7,6,9,4,3,1,2,5,4,5,3,8,
2,1,6,7,9,3,6,2,8,4,7,5,9,1,1,9,7,6,5,3,4,2,8,5,8,4,1,9,2,6,7,3,
4,7,9,5,3,1,2,8,6,6,1,3,2,7,8,9,5,4,8,2,5,4,6,9,3,1,7,9,4,1,7,2,
6,8,3,5,7,3,6,9,8,5,1,4,2,2,5,8,3,1,4,7,6,9]]
1
Picat> exit
```

Rychlost byla neuvěřitelná, u této úlohy hluboce pod jednu vteřinu na, řekněme, ne úplně rychlém notebooku (staříček ThinkPad T400).

6. Pár slov závěrem

Závěr naší úlohy by byl už jen čistě typografický, vysázet oněch 405 proměnných do formy olympijských kruhů. Nechávám případným badatelům na zkoumání. Důležité je, že existuje řešení, a to právě jedno, což jsme potřebovali dokázat a ověřit si.

Raylib + R = Raylib × R = RaylibR: AHOJ, SVĚTE!
Raylib + R = Raylib × R = RaylibR: HELLO, WORLD!

Pavel Stříž

E-mail: pavel@striz.cz

Abstrakt: Článek stručně odkazuje na instalaci Raylibu (prostředí pro 2D a 3D grafiku napsaný primárně v programovacím jazyce C), ale též na mnohem důležitější RaylibR. Jedná se o most mezi Raylibem a výpočetním prostředí R. Užívá k tomu eRkový balíček `Rcpp`. Byl to do určité míry boj, který však stál za to.

Klíčová slova: Raylib, R, RaylibR, Rcpp.

Abstract: The article briefly refers to an installation of Raylib (an environment for 2D and 3D graphics written primarily in the C programming language), but most importantly to an installation of RaylibR. That's a wrapper of the Raylib environment for the R environment written with the help of the `Rcpp` package. It was a bit of struggle, but it paid its dividends in the end.

Keywords: Raylib, R, RaylibR, Rcpp.

1. Jak jsem do toho spadl

Musím se přiznat, že nejsem Céčkař (nedejbůh Cépépéčkař, zatím) ani eRkař, aspoň se tak necítím, učím se za pochodu. Během programování sudoku s překryvy (angl. multi-sudoku puzzle; multi-grid sudoku; overlapping sudoku) jsem měl nápad udělat si 2D a 3D vizualizaci ze zadání do řešení. Ve stylu arkádovek 80. let, či aspoň nějak podobně. TEX umí hodně věcí, ale tohle není jeho doména, resp. má doména v TEXu, tak jsem hledal alternativu.

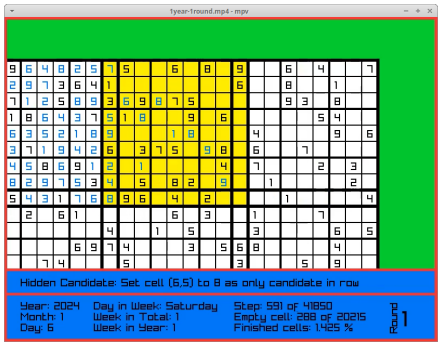
Rudolf Blaško má hezké výstupy přes Asymptote, ale na ten jsem rezignoval, potřeboval jsem ještě větší flexibilitu (myšleno programátorskou svobodu). Na vlastní prostředí jsem to také neviděl.

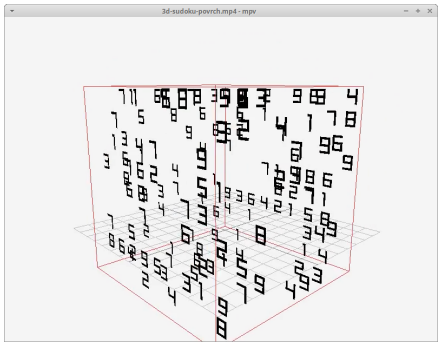
Zkusil jsem tedy řadu existujících grafických prostředí pro C, moc se mi líbí (od X11, FLTK, ImGui, GTK+ přes Qt, Cairo, SFML až po SDL2), objevil jsem i malé projekty typu Olive.c a v tom čase jsem narazil na Raylib od Ramona Santamarie. Užívá GLFW a OpenGL. Zrovna na YouTube slavil 10 let existence projektu. Objev byl učiněn hlavně díky tomuto videu https://www.youtube.com/watch?v=0To1aYglVHE, kde uživatel Tsoding Daily zkoumá uložení projektu do videa.

Prošel jsem webové stránky a zjistil, že by to mělo běžet na většině operačních systémů, platforem a ve všech známějších programovacích jazycích. Ani jsem nepomyslel na Python, hned jsem skočil po nativní verzi

v C. Za pomoci seznamu příkazů, https://www.raylib.com/cheatsheet/cheatsheet.html, a několika videí na YouTube, to začalo běžet.

Přikládám dvě ukázky. Není tam běhající či létající panáček či panenka, bylo potřeba se držet svých mantinelů. Sudoku generuje Lua, to jsou mé experimenty do Vánoce 2023, a Python3, to jsou nové pokusy od povánočních svátků dál. 2D rozkres krychle, viz zadání na str. 21 a řešení na str. 20.





2. Instalace knihovny Raylib

Co byl pro mne problém byla instalace, jak pracuji záměrně na starém notebooku, protože co mi běží na něm, poběží i na rychlejších strojích, toť základní idea. Hlavní zádrhel byl, že nemám OpenGL verzi 3.3, ale nižší (v2.1), tedy se to při instalaci knihovny a ukázek musí nastavovat. Svou verzi zjistíte na Linuxu přes knihovnu `mesa-utils`:

```
$ glxinfo | grep "OpenGL version"
```

V principu jsem následoval návod, přes `sudo make install` mám knihovny pro Desktop verzi, lokálně pak pro Web verzi (tu používám minimálně). Tím nenápadně naznačuji, že Raylib umí generovat své výstupy pro webové prohlížeče – za pomocí WebAssembly.

Když mé náhledy videoukázek viděl Aleš Kozubík vznesl dotaz, jestli by nebylo možné Raylib aktivovat v R, že by se jim tam něco takového hodilo. Na první dvě kola jsem R v seznamu jazyků na Raylibu na mobilu přehlédl, užil jsem RaylibRS, jak je v logu písmeno R, to bylo však pro Rust. Pak jsem si očistil brýle a na normálním monitoru to našel! Ve vyhledávači jsem se přímo zeptal na RaylibR.

3. Instalace RaylibR

Instalovat Raylib sólově netřeba, jen si stačí pobrat závislé balíčky.

```
sudo apt install build-essential git cmake libasound2-dev
  libx11-dev libxrandr-dev libxi-dev libgl1-mesa-dev
  libglu1-mesa-dev libxcursor-dev libxinerama-dev
```

Zkusil jsem v R:

```
> install.packages("Rcpp")
> install.packages("remotes")
> remotes::install_github("jeroenjanssens/raylibr")
```

Ale instalace mi zkolabovala, neb nemůže najít `R_ext/Error.h`. Začal mi trochu boj. Podezříval jsem `Rcpp`, ale nebylo to tím. Na GitHubu v *Open Issues* je tento problém jako jediný otevřený (k datu 17. 2. 2024), ale tipy na řešení mi nezabraly, tak jsem zkoumal. Našel jsem dvě cesty.

První, byť druhá v pořadí vzniku, je tato. Soubor mi v počítači existuje, jen v trochu jiné složce. Jestli je to dané tím, že jsem R instaloval z linuxového repozitáře versus přímo verzi ze CTANu, nevím. V `~/.bashrc` jsem si přidal:

```
export C_INCLUDE_PATH=$C_INCLUDE_PATH:/usr/share/R/include
```

Je to vlastně snadné, když člověk ví, co hledat. Má první cesta byla o něco bojovnější. Stáhl jsem si z GitHubu RaylibR. Rozbalil jsem `raylibr-main.zip`. Nic s termínem `Error.h` jsem nenašel. Po hlubším zkoumání jsem zjistil, že Raylib je přítomen ve složce `inst/` jako `raylib-4.0.0-modified.tar.gz`, rozbalil jsem jej, a pak už to bylo „snadné“. Ve složce `src/` je `Makefile`, do kterého jsem zasáhl. V mém případě na řádku 207 odkomentováním:

```
GRAPHICS = GRAPHICS_API_OPENGL_21
```

Ale co je důležitější, přidal jsem na řádku 379 na konec parametr na mou cestu s žádaným souborem, do tvaru:

```
INCLUDE_PATHS = -I. -Iexternal/glfw/include -Iexternal/glfw/deps/mingw
  -I$(R_HOME)/include -I/usr/share/R/include
```

Zabalil jsem složku `raylib-4.0.0-modified` do `tar.gz`. A chybí už jen jeden krok, jít o dvě úrovně výš a zabalit složku `raylibr-main` do souboru `raylibr-main.tar.gz`. Pozor, R (zatím) neumí pracovat se `zip` soubory.

V R už to pak bylo přímočaré:

```
> install.packages("raylibr-main.tar.gz",repo=NULL)
```

Radost to byla veliká, byť takto psané mi to přijde vše jasné, stručné a logické. Své dva objevy jsem připsal autorovi na GitHub, je velká šance, že i tato chyba bude uzavřena.

4. Několik ukázek

Seznam ukázek získáme v R přes:

```
> library(raylibr)
> demo(package="raylibr")
```

Zkusit si jednu z nich, *Hello, World!*, můžeme takto:

```
> demo("helloworld",package="raylibr") # nebo
> demo(raylibr::helloworld) # či dokonce jen demo(helloworld)
```

Alternativa zobrazení dostupných ukázek v R je:

```
> help.start()
  Vybrat: Packages --> raylibr --> Code demos.
```

Ukončení R je příkazem `q()` nebo `quit()`. Volíme ano (y) či ne (n), chceme-li si uložit pracovní prostředí, nechceme-li práci zatím přerušit volíme pokračovat (c).

V krátké přednášce na YouTube Jeroen Janssens, autor RaylibR, zmínil své dva vzorky – `raylibr::stroop` v úvodu povídání a `raylibr::beatbox` v jejím závěru.

A mj. ty ukázky jsou interaktivní, takže toho hada si můžete zahrát, v tom bludišti skutečně můžete chodit ap. Více ukázek (přes sto) hledejte přímo na stránkách Raylibu, v RaylibR jich je „jen“ jedenáct. Po instalaci jsem `*.R` našel v `~/R/x86_64-pc-linux-gnu-library/4.3/raylibr/demo/`.

Pro úplnost článku uvádím i zdrojový kód pro R.

```
library(raylibr)
init_window(600,400,"R & Raylib: Hello, World!")
while (!window_should_close()) {
  alpha<-abs(sin(get_time()))
  begin_drawing()
  clear_background("black")
  draw_circle(300,200,seq(150,10,by=-10),c("red","white"))
  draw_text(c("Hello,","World!"),225,c(120,220),64,fade("black",alpha))
  draw_fps(10,10)
  end_drawing()
  }
close_window()
```

Soubor se jmenuje `helloworld.R` a spustíme si jej z příkazového řádku přes:

```
$ Rscript helloworld.R
```

Okno s Raylibem se zavře přes klávesu Escape.

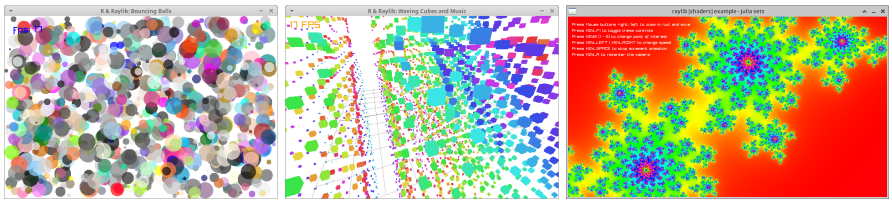
5. Závěrem

Asi by byla pro mne zajímavá výzva zkusit své „arkádové“ pokusy (Raylib+C) překlopit do Raylib+R a podat o zkušenostech s konverzí zprávu, ale nejdu do toho. Jsou další úkoly. Možná by šel udělat automat na konverzi, nechávám jako otevřený problém.

Musím se přiznat, že to byl pro mne boj, i Raylib i poté RaylibR, ale zvítězil jsem. Těším se, až nahodím silnější stroj s OpenGL v3.3 a zkusím si instalace ještě jednou. O tom možná pár slov na konferenci OSSConf, do termínu uzávěrky sborníku testy skoro určitě nestihnu. Hlavně se těším na ukázky v Raylibu (speciálně složka `shaders/`), ne všechny mi plně jely. Možná to bude tím, že pro v2.0 a v2.1 není podpora ve složce `examples/shaders/resources/shaders/`, ale jen pro starší OpenGL v1.0, v1.2, a pak až pro novější v3.3...

Držím s instalacemi palce, nenechte se odradit, pokud vám něco na prvních deset pokusů nejede!

Zde jsou mé oblíbené ukázky: `raylibr::balls` za 2D a `raylibr::cubes` za 3D grafiku, už běžící přes RaylibR.



6. Post Scriptum

Mohu shrnout zkušenosti z novějšího notebooku s Xubuntu a OpenGL v4.3. Instalace Raylibu byla vzorná dle návodu, předvolená je verze 3.2 OpenGL, nebylo tedy potřeba zasahovat. Po přidání zmíněného řádku do `~/.bashrc` mi šla i instalace RaylibR přímo v R. Poznámka: v pozadí RaylibR bere Raylib verze 4.0, je však již verze 5.0. Je stále co zlepšovat!

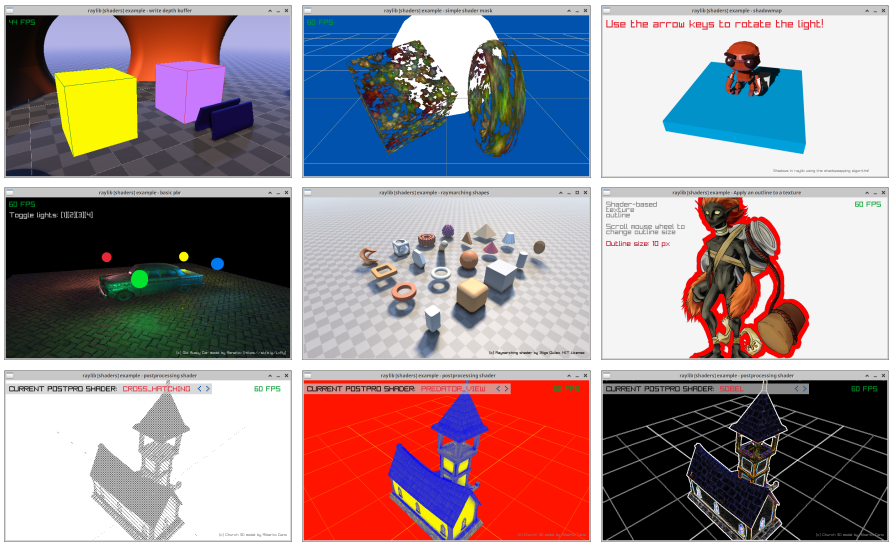
Zde je drobný dávkový soubor, který vám spustí všechny zkompilované ukázky, jednu za druhou. Napočítal jsem jich 146+1 šablona. Soubor mám uložený ve složce `raylib-master/`.

```
# Prázdný řetězec pro všechny, či vybranou složku z:
# audio core models others shaders shapes text textures
malkde=""
cd examples/$malkde
for soubor in ‘find . -type f -executable | sort‘; do
```

```
echo "Spouštím $soubor..."
./$soubor # tichý režim: ./$soubor 1>/dev/null 2>/dev/null
done
```

Ukázky jsou výtečné, minimálně animovaná `raylibr::julia` v R či ukázka `shaders_julia_set` v Raylibu je neokoukaná (obrázky výš, vpravo).

Pro potěšení oka, z těch dalších, které mi v Raylib v5.0 na stařičkovi nejely, zmíním ze složky `examples/shaders/`, prefix `shaders_`, tyto ukázky: `write_depth, simple_mask`, `shadowmap` na prvním řádku, `raymarching, basic_pbr, texture_outline` na 2. řádku a na 3. řádku z `postprocessing`, speciálně cross_hatching, predator_view a sobel.



7. Symbolická poznámka

Název článku má být „pomsta“ za ten boj s instalacemi. Při $x + y = x \cdot y$, tedy $y = x/(x-1)$ či $x = y/(y-1)$, ani jeden z programů nemůže být sám o sobě jedničkou. Raylib je skvělý na hry, ale o světě R neví nic. Naopak R má na vizualizaci tohoto typu ještě dost velké rezervy. Ale oba programy mohou být zároveň nula, to byl ten případ, kdy se nedařilo nainstalovat Raylib ani RaylibR, ze začátku to nešlo a nešlo…

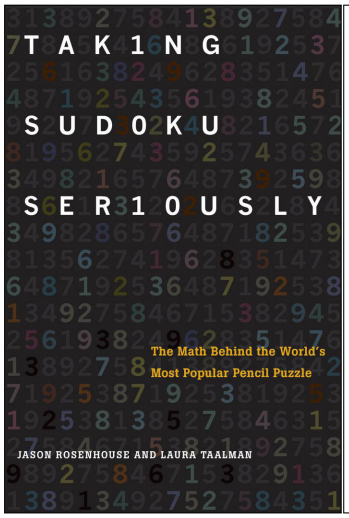
Držím palce, ať je vždy $x + y > 1$! Třeba dvě dvojky, nemusí to být hned jedničky s hvězdičkou.

RECENZE KNIHY: TAKING SUDOKU SERIOUSLY
BOOK REVIEW: TAKING SUDOKU SERIOUSLY

Pavel Stříž

E-mail: pavel@striz.cz

Jason Rosenhouse, Laura Taalman: *Taking Sudoku Seriously – The Math Behind the World's Most Popular Pencil Puzzle,* Oxford University Press, New York, New York, 2011. ISBN 978-0-19-975656-8.

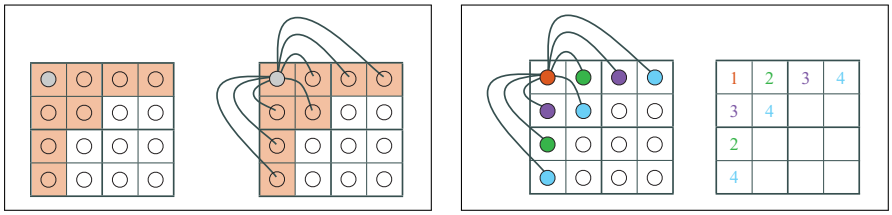


Motto knihy: *Sudoku is Math in the Small.* Kniha začíná z pohledu hráče popsáním základních herních strategií, včetně Ariadne's Thread, nebo-li volně česky metoda pokus-omyl (někdo to nazývá hádáním), tedy cílené hledání. Diskutují proč i tato metoda má patřit mezi základní herní strategie.

Latinské a řecko-latinské čtverce tvoří druhou a třetí kapitolu knihy. Pro knihu je typické, že ukazují principy a výpočty na malých rozměrech a pak se snaží o zobecnění. Čtvrtá a pátá kapitola patří výsledkům počtu různých řešení sudoku (Felgenhauer, Jarvis), včetně zohlednění rotace a symetrie (Russell, Jarvis).

Velkou práci si v 6. kapitole dali s hledáním rotačně symetrického sudoku s 18 nápovědami. Zmiňují své postupy, úspěchy a neúspěchy. V této kapitole též diskutují měření obtížnosti sudoku.

V 7. kapitole se zaměřují na propojení sudoku s teorií grafů (Graph Coloring, Book Embeddings). Vtípků je v knize celá řada, mně se líbil tzv. 7-legged spider, tedy sedminohý pavouk z této kapitoly, str. 129 a 130.



Řešení sudoku přes soustavu rovnic je náplň 8. kapitoly. Co jim zafungovalo u malého rozměru (Shidoku, sudoku velikosti 4×4), tedy zajištění cifer 1–4 právě jednou v součtu a součinu $w + x + y + z = 1 + 2 + 3 + 4 = 10$ a $wxyz = 1 \times 2 \times 3 \times 4$, už nefunguje u klasického sudoku, neb jsou v takové soustavě dvě řešení. Cifry 1–9 je jedno řešení, čtenáře se ptají na druhé. Došlo na připomenutí, že cifry 1–9 jsou v sudoku užity jen symbolicky. Tedy nepříjemnou situaci řeší jinou volbou cifer. Opět vyzývají čtenáře, ať si úlohu nalezení jiných 9 cifer prvně zkusí sami.

V 9. kapitole se zaměřují na rekordy v sudoku, chceme-li otevřené problémy. Kniha je již staršího data (2011), ale kromě dokázaného nutného minima 17 nápověd v sudoku na svém netratí. Zaujal mě problém sudoku s maximem nápověd, které má jen jedno řešení a nedá se žádná z nápověd odebrat (Maximum Independent Sudoku, nejvyšší známý počet je 39, sudoku č. 71, zadání na str. 159, řešení na str. 154), a sudoku, jehož řešení se nejčastěji vyskytuje v databázi sudoku s jen 17 nápovědami (The Strangely Familiar Sudoku Square, aktuální rekord je 29, spodní sudoku na str. 169). Za pozornost určitě stojí i minimum nápověd v sudoku X (aktuální rekord je 12 nápověd, sudoku č. 76, zadání na str. 166, řešení na str. 201).

V Epilogu, 10. kapitole, zahltili hráče, tedy čtenáře knihy, nespočtem různých dalších variant sudoku. Řešení sudoku a jejich variant je možné občas zahlédnout v textu, ale zmiňují to pod zadáním pro případného hráče, nebo až na konci knihy. V knize je zmíněno celkem 97 problémů, sudoku a jejich variant. Vyzývají čtenáře, aby si je zkusili. Řada variant je známých, některé méně. Nechtěl bych být jejich beta-tester, to byla Rebecca Fieldová, která je z hráčského pohledu zkusila všechny. Některá sudoku jsou totiž pětihvězdičková. Pro zájemce zmiňuji současný a aktivní kanál na YouTube *Cracking the Cryptic*, který se na různé varianty sudoku speciálně zaměřuje.

Mé oblíbené varianty sudoku s překryvy v knize zastoupené byly, např. Venn Sudoku (tři sudoku propojené ve tvaru Vennového diagramu, č. 56, zadání na str. 122, řešení na str. 199) a Samurai Sudoku X (pět překrývajících se sudoku ve tvaru X, ale každé sudoku navíc s podmínkou jednoho opakování všech cifer 1–9 v obou diagonálách, č. 95, zadání na str. 189, řešení na str. 205).

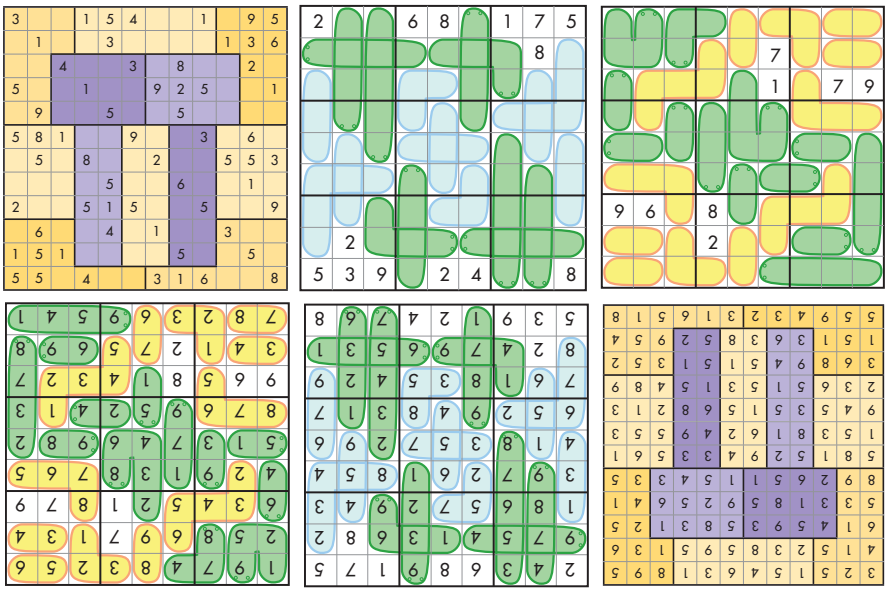
Za zvláštní pozornost jistě stojí i sudoku, kde bloky tvoří polomagické čtverce velikosti 3. Tedy magický čtverec, kde stejný součet 15 očekáváme

v řádcích a sloupcích, nikoliv však v diagonálách (Three-Magic Sudoku a All-Magic Sudoku, č. 85 a 86, zadání na str. 179 a 180, řešení na str. 203).

Knihu lze hezky shrnout jejich větičkou: *There is such a thing as a mathematical view of the world.* V závěru autoři zmiňují, že podkladů pro další knihu by měli dostatek. A ptají se čtenářů, jestli do vydání jít. Zatím jsem další jejich knihu nepotkal, asi její čas ještě nedozrál, nebo jsem málo hledal. V knize hodně odkazují na matematické termíny Gerechte Designs a Gröbner Bases, to stojí za bližší prozkoumání po dočtení knihy.

Kniha se dá přečíst jedním dechem za den, pro příznivce rekreační matematiky už, de facto, klasika. Je zaslouženě věnována Martinu Gardnerovi, průkopníku rekreační matematiky, jak by současná generace řekla o YouTuberovi: influencer, dokonce hned několika generací, včetně Donalda E. Knutha, viz např. jeho obří knihy TAOCP, polozkr. Tao of Computer Programming.

Kniha obsahuje řadu barevných sudoku, pro potěšení oka čtenáře bulletinu a hráčské vášně zmiňuji sudoku č. 57, 91 a 92: Jigsaw Pi Sudoku a Worms (Červíci). Vedle základních pravidel sudoku jsou zde podmínky: v prvním sudoku se v blocích opakují čísla z 3,14159265358; v zelených je rostoucí sekvence čísel od ocasu k hlavě, v modrých je ta podmínka stejná (jen prostřední obrázek), ale musí se navíc zjistit, která část je hlava a která ocas, a u žlutých (jen pravý obrázek) je rostoucí sekvence, ale rozdíl dvou sousedních buněk je vždy jedna. V knize str. 127, 185 a 186 (zadání) a str. 199 a 204 (řešení).

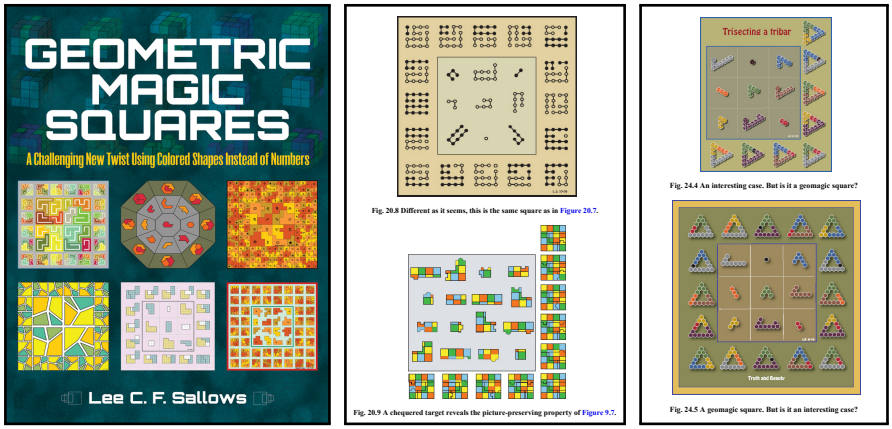


RECENZE KNIHY: GEOMETRICKÉ MAGICKÉ ČTVERCE
BOOK REVIEW: GEOMETRIC MAGIC SQUARES

Pavel Stříž

E-mail: pavel@striz.cz

Lee C. F. Sallows: *Geometric Magic Squares – A Challenging New Twist Using Colored Shapes Instead of Numbers.* Dover Publications, Inc., Mineola, New York, 2013, 245 pp. eISBN 978-0-486-29002-7. Vedle titulní strany jsou vidět ukázky ze stran 126 a 161.



Kniha se zabývá novátorským přístupem k magickým čtvercům. Skládají se geometrické obrazce, a to tak, že místo součtu čísel dostáváme jedinečný, ale stále stejný obrazec. A to obvykle po složení v řádcích, sloupcích a hlavní a vedlejší diagonále. Skládané geometrické objekty lze libovolně otáčet a překlápět. Kniha je jistým završením několika desítek let práce a experimentů započatých a postupně zveřejňovaných a diskutovaných na webové stránce autora, viz http://www.geomagicsquares.com.

Kniha by se dala hezky shrnout slovy autora, jehož palindrom – Lee Sallows swollas eel – ho doslova předurčil k napsání takové knihy:

> *Squared paper, I might add, along with a pencil and eraser (as well of course, as a magic wand) are the indispensible tools of the practicing geomagician.*

Základem je pro autora magický čtverec velikosti 3×3 (anglicky Magic Square, čínsky Lo shu), na kterém autor vysvětlování začíná, viz str. 12 (obrázek vlevo na další straně).

Kniha operuje s třemi hlavními principy práce, a to metodou pokus-omyl, nazvěme to tvůrčím experimentováním, poté užívá výpočetní techniku, píše o vlastních programech za pomoci programátora Pata Hamlyna (polyforms) a opírá se o Lucasovo pravidlo (anglicky Lucas's rule), volně upřesněno jako Lucasův mustr či Lucasova předloha. Toto algebraické pravidlo dále různě skloňuje, zobecňuje a rozšiřuje, viz str. 15 (obr. vpravo a níž vlevo aplikace).

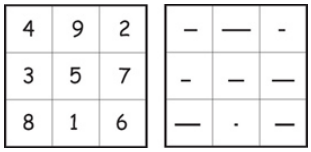


Fig. 2.1 A geometrical version of the *Lo shu*.

$c+a$	$c-a-b$	$c+b$
$c-a+b$	c	$c+a-b$
$c-b$	$c+a+b$	$c-a$

Fig. 2.3 Lucas's formula for the general 3×3 numagic square.

Autor rozdělil knihu do tří velkých částí: geomagické čtverce 3×3, geomagické čtverce 4×4 a speciální kategorie, řekněme, kategorie ostatních. Velkou část knihy tvoří 5 příloh, rejstřík a literatura. Je psána poeticky a obsahuje řadu (matematických či geometrických) vtípků. Dovolím si je neprozrazovat. Autor zavádí řadu nových termínů a poukazuje i na slepá místa svých bádání, konkrétně zmiňuje otevřené problémy a užší oblasti.

Badatele v této oblasti určitě potěší rozbory komplikovanějších partií, včetně geomagických čtverců ve 3D, skládání dílků částečně či úplně oddělených či seskládání hezkého obrazce s jedním či dvěma otvory. Poukazuje i na výstupy svých kolegů-badatelů. Osobně mě zaujala autorova úvaha nad výstupem programu, který sice splnil autorovo zadání (složení kostky), ale kostka je v našem fyzickém světě nerozebíratelná, viz str. 133 (obr. vpravo).

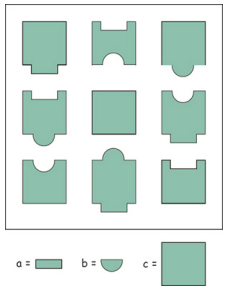


Fig. 2.4 Lucas's formula realised in geometric shapes.

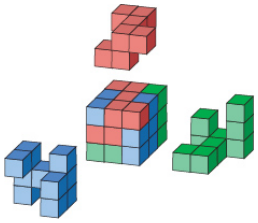


Fig. 21.3 A cube that cannot be dismantled.

Kniha neobsahuje zdrojové kódy, o to víc to láká čtenáře sednout k počítači a mrknout se, kdo, co a kde tvoří. Kniha se dá přečíst za den bez větších problémů, ovšem její úplné podchycení, to je tak na dva roky.

Kdo má rád rekreační matematiku, latinské a řecko-latinské čtverce, problémy podobné sudoku a skládání puzzle, bádání nad symetrií a programováním, tuto knihu by měl čtenář minimálně prolistovat jako zdroj inspirace.

REŠERŠE ZDROJŮ: LOGICKÁ HRA SUDOKU
RESEARCH OF RESOURCES: THE SUDOKU PUZZLE

Pavel Stříž

E-mail: pavel@striz.cz

Články

Rád bych upozornil na tři články:

- Arnab Kumar Maji et al.: An Exhaustive Study on Different Sudoku Solving Techniques. *International Journal of Computer Science Issues*, Vol. 11, Issue 2, No. 1, March 2014. ISSN 1694-0814, eISSN 1694-0784. https://www.ijcsi.org/papers/IJCSI-11-2-1-247-253.pdf
- Mária Ercsey-Ravasz, Zoltán Toroczkai: *The Chaos Within Sudoku,* arXiv: 1208.0370v1, August 1, 2012. https://arxiv.org/pdf/1208.0370.pdf
- Gary McGuire, Bastian Tugemann, Gilles Civario: *There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration,* arXiv 1201.0749v2, August 31, 2013. https://arxiv.org/pdf/1201.0749.pdf

Knihy

- Dvě knihy byly zmíněny. *Taking Sudoku Seriously,* viz str. 32, a,
- *Geometric Magic Squares*, viz str. 35.
- Wei-Meng Lee: *Programing Sudoku,* Apress, USA, 2006. ISBN 978-1-59059-662-3. Zdrojové kódy jsou na https://github.com/Apress/programming-sudoku.
- Giulio Zambon: *Sudoku Programming with C,* Apress, USA, 2015. ISBN 978-1-4842-0996-7. Zdrojové kódy jsou dostupné na https://github.com/apress/sudoku-programming-w-c.

Efektivní algoritmus

Efektivní algoritmus vyvinul Donald E. Knuth známý jako DLX, Dancing Links, Algorithm X. Knuth, Donald E. (2000). Dancing links. *Millennial Perspectives in Computer Science.* P159. 187. arXiv 0011047v1, November 15, 2000. https://arxiv.org/pdf/cs/0011047.pdf

Volně dostupné programy

V Linuxu jsou k dispozici balíky:

- `sgt-puzzles`, příkaz `sgt-solo`.
- `qqwing`, příkaz ten stejný.
  Umí vypsat kroky u řešení: `qqwing --generate 1 --instructions`.
- `gnome-sudoku`, příkaz stejný.
- `nbsdgames`, příkaz `nbsudoku`. Tip: `nbsudoku -s 7`.
- `sudoku`, příkaz stejný.
- `ksudoku`, příkaz stejný. Umí řadu typů, včetně 3D.
- `fltk1.3-games`, příkaz `flsudoku`.

Program Sugen byl zmíněn v článku od str. 16, viz https://dlbeer.co.nz/articles/sudoku.html.

Na GitHubu má uživatel KyleGough svůj program `sudoku`, který řeší sudoku pomocí logických, resp. hráčských metod, viz https://github.com/KyleGough/sudoku.

Další zajímavé zdroje

Počty různých sudoku, viz http://www.afjarvis.org.uk/sudoku/.

Řešení různých variant sudoku, viz kanál na YouTube https://www.youtube.com/@CrackingTheCryptic.

Guinnessova kniha rekordů, https://www.guinnessworldrecords.com/. Zajímavé a aktivní jsou tři: Largest multi-sudoku puzzle, Most people playing sudoku simultaneously a Biggest sudoku published (jedná se o sudoku 100 na 100).

NOVÉ A AKTUALIZOVANÉ BALÍČKY V TEXOVÉM SVĚTĚ
NEW AND UPDATED PACKAGES IN THE TEX WORLD

Pavel Stříž

E-mail: pavel@striz.cz

Abstrakt: Článek ve zkratce představuje nové a nově aktualizované TEXové balíčky z úplné distribuce TEXLive 2023. V úvodu zmiňuje ~~špinavý~~ užitečný trik, jak si aktualizaci balíčků seřadit dle tzv. revize, aby si usnadnil prohledávání. Rozhodně to není univerzálně přijatelná metoda, ale byla to taková nouzová varianta, která autorovi, v současnosti bez internetového připojení, zafungovala. I tak nechť čtenář bere prosím na vědomí, že se jedná o subjektivní výběr.

Klíčová slova: TEX, TEXLive.

Abstract: The article briefly introduces new and recently updated TEX packages presented in the TEXLive-full distribution from 2023. The author presents a ~~dirty~~ nifty trick of sorting revisions of the TEXLive repository to easier the process of searching them. It is definitely not a genuine and recommended way of doing things, but the best the author could find on his notebook currently without the internet connection. Please be aware, the selection of presented packages is still a personal and subjective preference of the author.

Keywords: TEX, TEXLive.

1. Hledání novinek

Když občas zatoužím podívat se po nových TEXových balíčcích, nedělám to většinou moc systematicky. Podívám se na `tug.org` do novinek, když je víc času nahlédnu na jejich mailinglist.

Častěji spíš aktualizuji TEXLive a podívám se do logu, přesněji po instalaci do souboru `texlive/[rok]/install-tl.log`, co mám na stroji nového.

```
$ tlmgr update --self --all
```

Většinou jsem prošel názvy balíčků a podíval se na dokumentaci některých, spíš náhodně. Když nebyl čas, tak jsem po aktualizaci nezvládl ani to. To mi, jako šéfovi sekce, s úsměvem vytkl Aleš Kozubík na posledním ročníku OSSConf, že by to chtělo širší rozhled, nedělat to náhodně a jít na to systematicky, neb jsem přehlédl balíček... Má pravdu, před mnoha lety jsem přehlédl beamer, tikz i pgfplots, narazil jsem na ně *jinou cestou*.

Relativně nová záležitost je výpis balíčků, tedy soubor `texlive/[rok]/doc.html`, ale má jednu závadu, je setříděn abecedně a není tam informace

o datech aktualizace. Přiznávám se, nevím, jaká je nejlepší/nejkratší cesta, následuje to, co jsem zkusil.

2. Python3 na pomoc

Poněvadž v době psaní tohoto článku nemám internet, musí si člověk vystačit s tím, co je po ruce. Pokud nahlédneme podrobněji do složky `texlive/[rok]/tlpkg` najdeme tam užitečné informace. Například ve složce `tlpobj` jsou metadata balíčků. Když se rozhlédneme ještě lépe, zjistíme, že vše máme v souboru `texlive.tlpdb.main.[hash]`. Vidíme tam něco takového.

```
name 00texlive.config
category TLCore
revision 54074
shortdesc TeX Live network archive option settings
longdesc This package contains configuration options for the TeX Live
[...]
```

Nejsou k dispozici data aktualizací, ale nepotřebujeme žádné zázraky. Tím nenápadně naznačuji, že nebudu procházet příkaz `\date` z dokumentace balíčků. Číslo revize, evidentně rostoucí posloupnost jisté identifikace, by nám mohlo stačit.

Připravil jsem si drobný program v Pythonu3 a po spuštění dostáváme výpis zmíněný níže.

```
import os
from collections import defaultdict # pydoc3 collections
data=defaultdict(); dleRevize=defaultdict(); citac=0
nazev="/home/malipivo/texlive/2023/tlpkg/" \
    "texlive.tlpdb.main.18022ca66fa110fb133d12433144df42"
soubor=open(nazev) # Otevření souboru...
while True: # Načítej řádek po řádku, dokud to lze.
    radek=soubor.readline()
    if not radek: break # Konec while, jinak si ukládej do dat.
    if radek[:4]=="name":
        citac+=1; data[citac]=defaultdict()
        data[citac]["name"]=radek[5:-1]
    if radek[:8]=="category":
        data[citac]["category"]=radek[9:-1]
    if radek[:8]=="revision":
        data[citac]["revision"]=int(radek[9:-1])
soubor.close()
# Rychlé vytažení jen balíčků; příprava na setřídění.
for d in data:
    if data[d]["category"]=="Package":
        dleRevize[data[d]["revision"]]=[data[d]["name"]]
# Setříděné dle nejnovějšího vypiš do terminálu.
dleRevizeSorted=sorted(dleRevize,reverse=True)
```

```
hranice=0; okolik=5; citac=0
for d in dleRevizeSorted:
    print(citac, d, dleRevize[d][0]); citac+=1
    if citac>hranice and citac<=hranice+okolik:
        os.popen("texdoc "+dleRevize[d][0])
```

Samozřejmě, pro účely článku, zmiňuji jen prvních několik řádků. Nazvěme to pracovně *hitparáda revizí*. Prvních pět balíčků se otevře přes `texdoc`. Přes proměnné `hranice` a `okolik` si můžeme zobrazování na elementární úrovni řídit či úplně vypnout. Proměnnou `nazev` si prosím upravte dle potřeby.

```
67868 wordcloud
67867 lualinalg
67866 writeongrid
67865 verbatimbox
67864 proflycee
67863 huawei
67862 arraycols
[...]
```

Určitě jde program udělat hezčí, rychlejší a efektivnější, nechám si poradit. Nyní však máme potřebné. Máme seznam revizí distribuce TEXLive setříděný dle nejnovějších. Teď už jen stačí si je procházet, případně přes příkaz `texdoc [balíček]` si příslušnou dokumentaci otevřít a bádat detailněji.

3. Novinky v grafice, PlainTEXu a LATEXu

A já bádal. V dalších kapitolách článku následují subjektivně vybrané nové balíčky či nové partie balíčků. Záměrně neuvádím ukázky, jednak ať čtenář nahlédne doma svými silami, a možná se mezi organizátory domluvíme a uděláme něco jako mapu novinek v papírové podobě na rozdávání. Možná si neodpustím vzorek či dva...

3.1. TikZ

Za pozornost na úvod jistě stojí balíček visualtikz, je to jistý náhled na to, co Ti*k*Z umí (obdoba balíčku visualpstricks pro PSTricks). Ovšem, když si ten čas dáte, projít tu a tam dokumentaci Ti*k*Zu není na škodu. Jsou tam tři velké nové kapitoly na animování přímo z Ti*k*Zu. Autor píše, že na animaci v PDF rezignuje, že to zkouší přímo do SVG. Zmiňuje možnosti přes SMIL, CSS a JavaScript, s tím, že zkusil SMIL. Z dokumentace jsem si vytáhl ukázku:

```
\documentclass[dvisvgm]{article}
\usepackage{tikz}
\usetikzlibrary{animations}
\begin{document}
```

```
\begin{tikzpicture} [flapping seagull/.pic={
\draw (0,0) :path={
    0s= {"{(180:3mm) to [bend left] (0,0) to [bend left] (0:3mm)}"=base},
    1s= "{(160:3mm) to [bend left] (0,0) to [bend left] (20:3mm)}",
    2s= "{(180:3mm) to [bend left] (0,0) to [bend left] (0:3mm)}",
    repeats};
}]
\pic :rotate={0s="0", 20s="90"} {flapping seagull};
\pic at (1.5,1.5) {flapping seagull};
\end{tikzpicture}
\end{document}
```

A po spuštění a otevření SVG na mě v levém horním rohu mávala křídla racků.

```
lualatex --output-format=dvi test-animace.tex
dvisvgm test-animace.dvi
firefox test-animace.svg
```

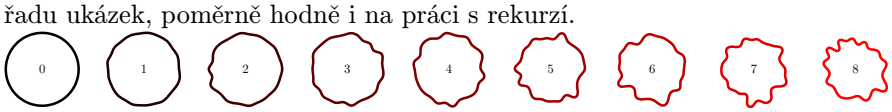
Dokumentace dále zmiňuje, že do PDF se dá vložit tzv. snapshot (připodobnění ke screenshot u monitoru), tedy náhled z animace v určitém čase. Dlouhodobě užitečné, řekl bych. Ještě zmíním, že autor Ti*k*Zu neustále pracuje na užití Lua na automatizované zobrazení grafů, tedy kdo ještě nepřešel na LuaLaTeX, měl by uvážit.

Kdo má rád Ti*k*Z braný do extrémů. Nechť nahlédne na balíček tcolorbox či na první stranu balíčku etoc. Zkratku etoc bychom mohli dekódovat jako „vysázení osnovy s efekty“ či volně jako „Extreme Table of Contents“.

Odpovědi na otázky na TeX.SE si uživatel Qrrbrbirlbel ukládá do knihovny tikz-extensions.

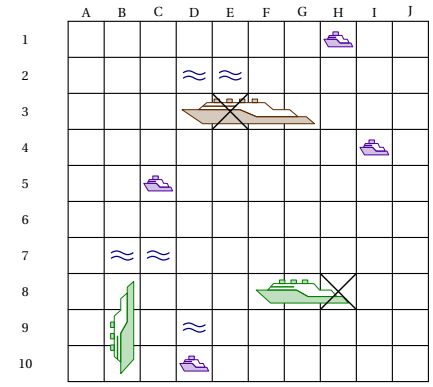
3.2. METAPOST

Za pozornost stojí neustálé bádání v METAPOSTu. Ať už by šlo o MetaFun v ConTeXtu (co tam Hans Hagen tvoří, z toho jde místy hlava kolem), nebo i pro LaTeXisty. Zmiňuji balíček drawing-with-metapost, který obsahuje celou řadu ukázek, poměrně hodně i na práci s rekurzí.

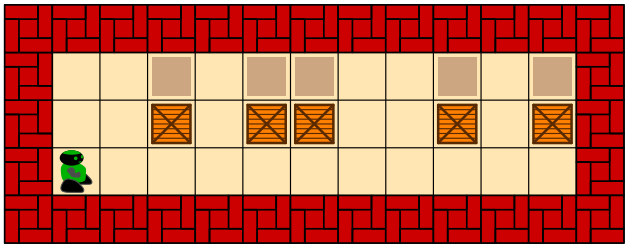


Velká část mého typografického bádání leží právě v METAPOSTu, protože vedle programu FontForge se přímo k Bézierovým křivkám těžko dostává. Mám otevřený problém, tzv. Wordcloud sázený TeXem. Dostávám se tímto k novému balíčku wordcloud, ale i tam dochází k závěru, že výpočet průsečíků křivek je neúnosně časově náročný. A pracovat s rastrem i vektorem současně: to zas není ještě úplně doména TeXu. Je to otevřený problém. Zde je jistá ukázka (obrázek levý).

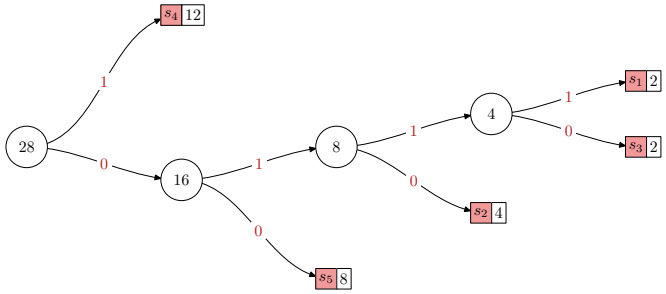
```
% lualatex vzorek-wordcloud.tex
\documentclass{article}
\pagestyle{empty}
\usepackage{xcolor,wordcloud}
\begin{document}
\wordcloud[scale=1, rotate=45, margin=0.5pt, usecolor,
    colors={red!40,blue!40,green!20!black}]{(OSSConf,10);
    (GIS,6);(Linux,7);(Lua,4);(Python,7);(TikZ,5);(R,8);
    (C,11);(JavaScript,4);(Žilina,10);(FŘI,5)}
\end{document}
```



Do třetice, zde je herní vzorek z knihovny repere (nahoře vpravo a níž).



Do n-tice zmíním ještě knihovnu huffman.



Poznámka. Uživatelé R dobře znají sadu písem Hershey. Zde se k nim dostaneme přes knihovnu hershey.

3.3. Lua, Python

Potěšil mě balíček piton, který přes Lua (LPeg) umí vysázet zdrojové kódy jazyků Python, C a OCaml. To není na škodu. Ale hlavně je to inspirativní, LPeg je silný nástroj, jen není jednoduché se do něj dostat.

Za nemalou pozornost stojí volání výpočetních prostředí přímo z TEXu, zaujal mě nový balíček luacas a aktualizované balíčky sympycalc a FenetreCas.

Za zvláštní pozornost, obzvlášť pro Rudolfa Blaška, jak tyto výrazy hodně sází, stojí balíček numerica, který za pomoci příkazu `\eval` sází matematický výraz zadaný TEXově, ale zároveň jej i spočítá.

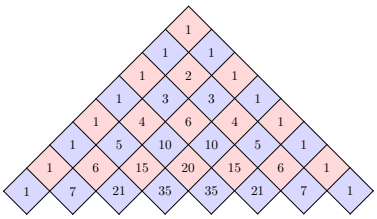
```
\eval[p=.]{\[
    \sum_{n=0}^{\infty}\binom{\alpha}{n}x^{n}
    \]}[\alpha=4.321,x=0.1234]
```

$$\sum_{n=0}^{\infty}\binom{\alpha}{n}x^{n} = 1.653329, \qquad (\alpha = 4.321, x = 0.1234).$$

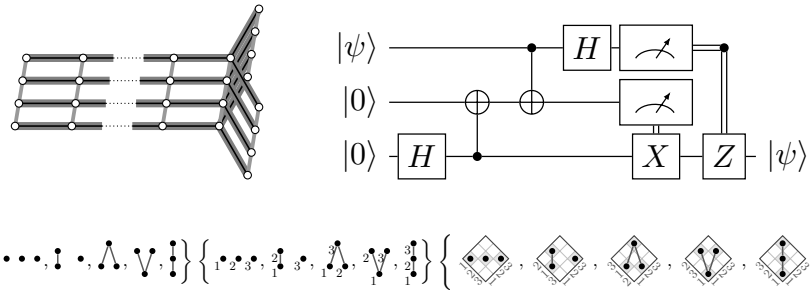
3.4. Matematika, fyzika, chemie

Na „ošipkování“ matematických vztahů zkuste balíček annotate-equations či witharrows.

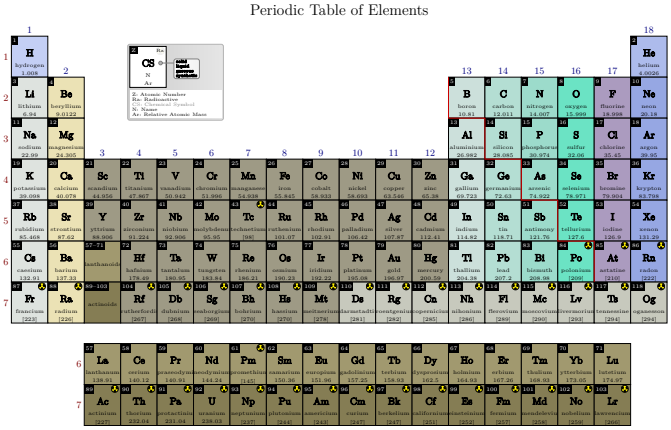
Sazba tabulek je v TEXovém světě věčné téma, pořád to není ono. Proto upozorňuji na balíček nicematrix (v ukázce s balíčkem adjustbox), který to zkouší po svém zas trochu jinak.



K tvorbě diagramů a stromů mě zaujaly balíčky dynkin-diagrams, yquant a causets. Nejsou to samozřejmě balíčky na denní užití, ale je dobré o nich vědět.



Co jsem chtěl vždycky umět je si vysázet periodickou tabulku prvků. Načíst si odněkud data, pohrát si s tím… To tým lidí stálo, pardón, to autora stálo neskutečné množství práce. Dívám se na dokumentaci balíčku pgf-PeriodicTable s otevřenou hubou. Poznámka: při načítání balíčku musí být dodržena malá a velká písmena. Zatím chybí české a slovenské popisky.



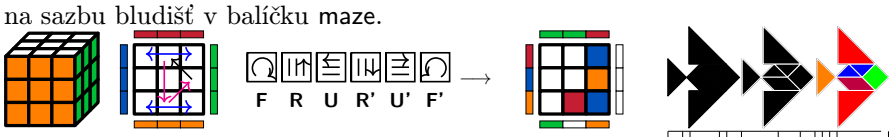
3.5. Rejstříky

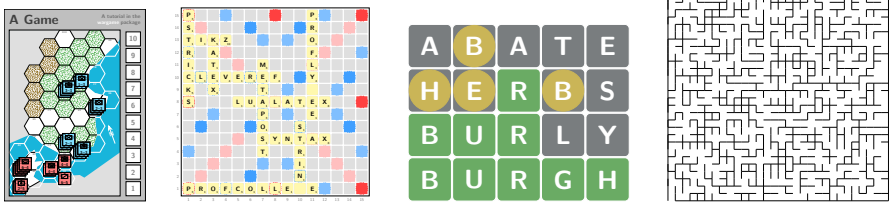
Sazba rejstříků je též jedno z velkých typografických témat. Herbertu Vossovi se v balíčku xindex už slušně daří držet pravidel UCA (Unicode Collation Algorithm). Dle počtu jazyků asi ještě nemůže být spokojen, ale daří se mu. Což o to, že už to češtinu umí, ale slovenština chybí! To tak dělat nemůžeme!

3.6. Hry

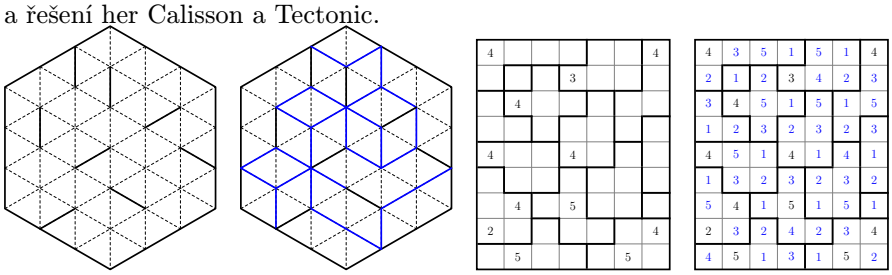
Zabývám se rekreační matematikou/kombinatorikou, většinou si výstupy sázím sám, ale stále sleduji co a kde vzniká, pro strýčka Příhodu.

Zde je soupis balíčků, které ještě nedávno neexistovaly. Jsou to rubikcube (zapínáme si `--shell-escape`), TangramTikz, wargame (ve spolupráci s balíčkem milsymb a balíčkem game ve složce jejich dokumentace; v této souvislosti dávám na pozornost i balíček hexboard), Scrabble, wordle či třeba jistý nástřel na sazbu bludišť v balíčku maze.





Zvláštní příspěvek, zde alespoň vlastní odstavec, si zaslouží balíček ProfCollege. Obsahuje celou řadu pomůcek k sazbě školních pomůcek. Na stranách 305–459 z 506 (!) lze nalézt hry, to množství je k nevíře. Zde je ukázka zadání a řešení her Calisson a Tectonic.



3.7. Náhled na písmo

Možná ti zkušenější znají, jak bylo a je možné si udělat rychlý náhled na sedmi- a osmibitová písma.

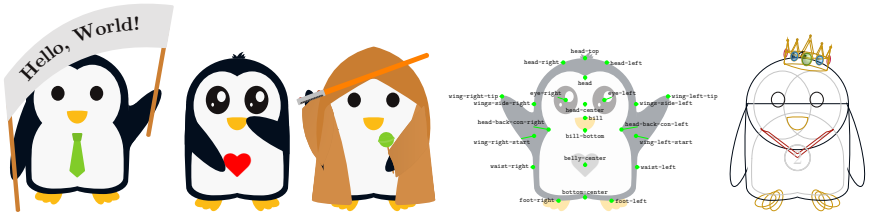
```
$ pdftex testfont
Name of the font to test = csr10 Enter
*\bye Enter
```

Kdo chce získat glyfy z TTF či OTF, může nyní využít balíčku unicode-fonttable, nemusí si to přes cyklus programovat v XeLaTeXu či LuaLaTeXu.

3.8. Symboly

Má oblíbená oblast jsou značky a symboly, těch není nikdy dost.

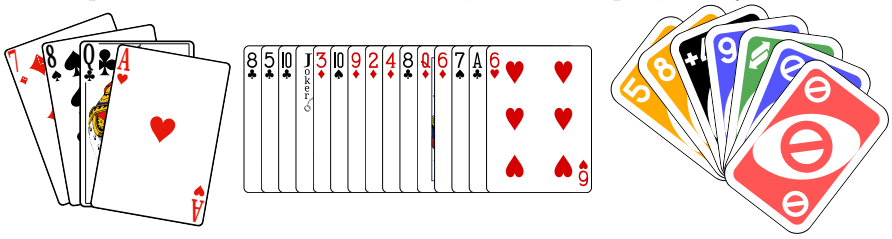
Zaujal balíček tikzpingus, ze zřejmých ping-us důvodů. Kdo neví, jak vypadá anatomie Tuxe, pak je to k nahlédnutí ideální balíček. Jen se mi na první dobrou nepodařilo vysázet srdce. Na druhou dobrou již ano, je potřeba načíst balíček fontawesome.



Co se mi však nepodařilo, bylo hezky a kompletně přesázet dokumentaci tohoto balíčku. Vypadá to, že přesázet si dokumentaci některých balíčků je slušné cvičení. Neurazí ani balíček tikzpeople.

Nevýhoda symbolů kreslených přes Ti*k*Z je, že se sazba dokumentu zpomaluje. Za pozornost proto dávám balíček simpleicons, to je sazba symbolů přes písmo. Autor sice píše, že je vývoj v alfa fázi, ale mně to přijde hodně slušné a funkční. Dost symbolů na vysázení kalendáře. Moment, jestli počítám dobře, tak v balíčku je 2625 symbolů, to bude víc jak na jeden kalendářní rok.

Má omezená slovní zásoba francouzských slov slavila úspěch, zvládl jsem si přeložit název balíčku JeuxCartes jako karty na hraní, dokumentace příjemně překvapila hezkou sadou karet nejen francouzského typu. Opět pozor na malá a velká písmenka v názvu balíčku. Autor, Cédric Pierquet, to rozjel ve velkém!



Hodně je vidět práce na pixelových obrázcích, namátkou balíčky pixelart, pxpic či kapitola 71 v balíčku ProfCollege (či přímo balíček PixelArtTikz).

3.9. Od místních

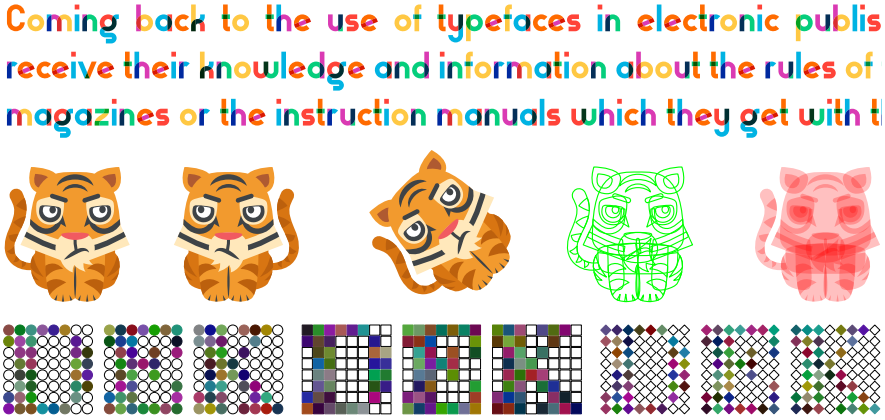
Od našich TEXistů-vývojářů jsem zahlédl aktualizace balíčků tex4ebook, lua-xml, biblatex-iso690, odsfile, linebreaker, ctanbib, make4ht, lua-uca (vše Michal

Hoftich), optex (Petr Olšák), pdfextra (Michal Vlasák), luavlna (Michal Hoftich, Miro Hrončok), zwpagelayout (Zdeněk Wagner), markdown a lt3luabridge (Vít Novotný). Díky Vám!

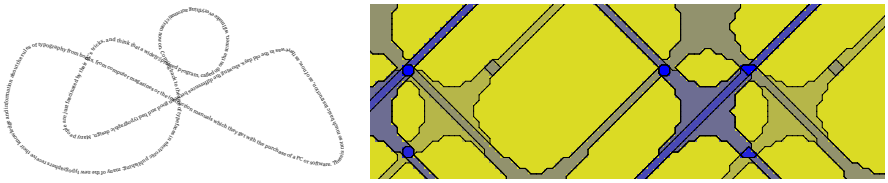
4. Novinky v ConTEXtu

ConTEXt nepoužívám na denní bázi, tam nahlížím po instalaci TEXLive do adresáře `texlive/[rok]/texmf-dist/doc/context`.

Vybírám několik ukázek z dokumentu `evenmore.pdf`, str. 15 (práce s písmy Type3), str. 16 (zobrazení SVG) a str. 22 (hrátky s písmem FONT36).



A několik z dokumentu `luametafun.pdf`, str. 15 a 31.



Kdo by se chtěl podívat na víc grafických ukázek, prolistujte si soubory `metafun-p.pdf` a `onandon.pdf`. Upozorním i na první zdařilé experimenty vložení SVG, viz `svg-lmtx.pdf`. Kdo by rád nahlédl na možnosti MathML, podívejte se na `mml*.pdf`.

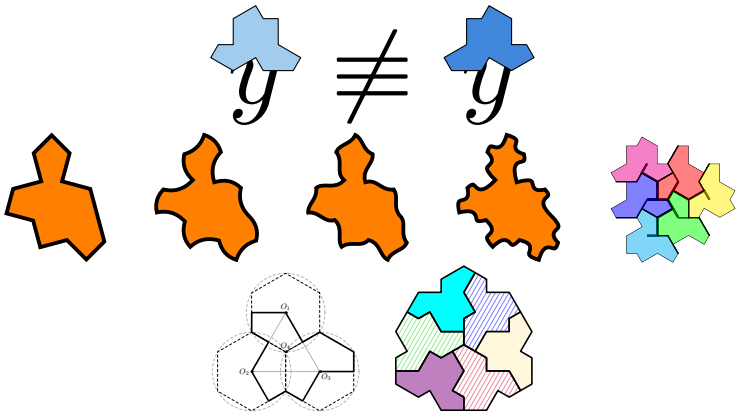
5. Místo Závěru jeden Einstein

Asi je to již známé, byl v březnu 2023 vyřešen tzv. Einstein/Monotile problém, jeden dílek, který pokryje plochu bez periodického opakování. Tzv. Hat se musel ještě zrcadlově překlápět, to autoři završili v květnu 2023 nálezem

tzv. dílu Spectre, který už nemusí vůbec nic. Důkazy jsou na pročtení hodny nejednoho matematika. Problém řešili mj. Roger Penrose i Terence Tao, ale nakonec jej zvládl amatér, Angličan David Smith. Potřebné matematické důkazy a počítačové programy, to je věc jiná, s tím museli pomoci profesionálové v oboru. U prvního článku lze stáhnout i kódy pro Python3. Upozorňuji, že důkazy jsou tam dva: kombinatorický a geometrický. Ten první zavádí novou, dosud neužitou formu matematického důkazu! Tohle vchází do dějin. Klobouk dolů všem!

Osobní poznámka. Za nemalou pozornost stojí, že se dají stáhnout LaTeXové kódy článků samotných. Inspirativní je i jejich `Makefile`. Studium souboru `Makefile` z jednoho balíčku (byl to přímo lshort-english) nám, Michalu Mádrovi a mně, pomohlo tehdy dostat lshort-czech na https://ctan.org/. Je to však již mnoho let, chtělo by to aktualizovat (a dostat se do hitparády revizí na první místo, vytlačit wordcloud, dodávám s úsměvem),...

Pro nás je však důležité, že už existují balíčky, kterými dílky můžeme sázet, jedná se o realhats, tilings, a nalezneme je, ještě jeden zápis pro METAPOST a upozornění na malá a velká písmena v názvu, i v obřím balíku – to už není balíček – ProfCollege. Klobouk dolů ještě jeden či dva!



6. Post Scriptum

Nedalo mi to a prošel jsem novinky i za červenec 2023 – polovina února 2024. Došlo na 483 revizí, nejnovější pod číslem 69933.

Už jen telegraficky. Robert Mařík upravil fancytooltips, Michal Hoftich zveřejnil responsive. Zaujaly mě balíčky circuitikz, numerica, tabularray, wrapfig2, keyfloat, zx-calculus, braids, pmdraw, wheelchart, onedown, TrivialPursuit, rank-2-roots a PanneauxRoute.

REŠERŠE ZDROJŮ: NEPERIODICKÝ DÍLEK
RESEARCH OF RESOURCES: AN APERIODIC MONOTILE

Pavel Stříž

E-mail: pavel@striz.cz

Dva klíčové články

- David Smith, Joseph Samuel Myers, Craig S. Kaplan, Chaim Goodman-Strauss: *An aperiodic monotile,* arXiv 2303.10798v2, March 2023, 89 pp. Článek obsahuje i zdrojové kódy pro Python. Dílek musí být ještě překlopen, což někteří kritizovali, že na ploše (2D) zrealizovat nejde. Dílek autoři nazvali The Hat. Lidově klobouček (tričko či kalhoty). https://arxiv.org/pdf/2303.10798.pdf
- David Smith, Joseph Samuel Myers, Craig S. Kaplan, Chaim Goodman-Strauss: *A chiral aperiodic monotile,* arXiv 2305.17743v1, May 28, 2023, 23 pp. Dílek, resp. celá rodina dílků, nemusí být ani překlápěna. Dílek nazvali The Turtle a The Spectre. Lidově želvička (může a nemusí se překlápět) a přízrak nebo vampír (nesmí se překlopit skrz oblení hran). https://arxiv.org/pdf/2305.17743.pdf

Další zdroje

Ze zdrojů na YouTube zmíním Craig Kaplan: Discovery of the Aperiodic Monotile na Numberphile, https://www.youtube.com/watch?v=_ZS3Oqg1AX0.

Jak dílky vysázet pomocí TEXových balíčků bylo zmíněno na předchozí straně. Lze očekávat ještě balíček, který vznik (nad)struktur ještě víc zjednodušší. První pokusy jsou trochu kostrbaté. V mezidobí lze využít webové stránky autorů a tam si rozkres uložit do png či svg, https://cs.uwaterloo.ca/~csk/hat/app.html, či využijte různé formáty z https://github.com/christianp/aperiodic-monotile.

Na testování skládání dílků existuje víc serverů, např. PolySolver, https://www.jaapsch.net/puzzles/polysolver.htm.