



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Numerické výpočty – IV

Jiří Zelinka

16. října 2013

Tento učební text vznikl za přispění Evropského sociálního fondu a státního rozpočtu ČR prostřednictvím Operačního programu Vzdělávání pro konkurenceschopnost v rámci projektu Univerzitní výuka matematiky v měnícím se světě (CZ.1.07/2.2.00/15.0203).

Obsah

1	Algoritmy teorie čísel	4
2	Algebraické výpočty	12
3	Kombinatorické a grafové algoritmy	23

Úvod

V tomto textu se ke slovu dostane zejména Sage, který jakožto symbolický nástroj oplývá mnohými algebraickými dovednostmi. Proto také si ve většině případů budeme toliko ukazovat, jak využívat již hotové nástroje. Ukážeme si také PARI/GP, který je patrně nejlepším výpočetním prostředkem pro algoritmy teorie čísel. A nezapomeneme na kombinatorické a grafové algoritmy. Pro zájemce o další zdroje uveďme alespoň [2] nebo [1].

Kapitola 1

Algoritmy teorie čísel

1.1 Prvočísla

Jednou ze základních záležitostí v teorii čísel je určení, zda dané číslo je prvočíslo, případně najít jeho rozklad na prvočinitele. To zvládne Sage celkem bez problémů, Matlab má zde jistá omezení:

```
sage: is_prime(2^16+1)
True
sage: is_prime(2^256+1)
False
sage:
```

```
>> isprime(2^16+1)
ans =
     1
>> isprime(2^256+1)
Error using isprime (line 24)
The maximum value of X allowed is 2^32.

>>
```

V Sage také můžeme zjistit nejbližší vyšší prvočíslo:

```
sage: next_prime(2^256+1)
1157920892373161954235709850086879078532699846656405640\
```


Zdá se, že s tím Sage nemá problémy. Kdybychom ale zkusil b rozložit na prvočísla, výsledku bychom se nedočkali, i když to, že b není prvočíslu se zjistí okamžitě:

```
sage: time is_prime(b)
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.00 s
False
sage:
```

1.2 Zbytkové třídy

Počítání a operace ve zbytkových třídách se objevuje jak v algebře tak i v teorii čísel. V Sage máme několik možností, jak výpočty zbytkové třídy modulo n provádět. Můžeme používat přímo funkci `mod`, nebo si definovat zbytkovou třídu jako objekt a pracovat s ní:

```
sage: x=mod(20,12)
sage: x
8
sage: x^10
4
sage: Z12=Integers(12)
sage: Z12
Ring of integers modulo 12
sage: y=Z12(20)
sage: y
8
sage: y^10
4
sage: x==y
True
sage:
```

Dostali jsme různým způsobem dva totožné objekty, což není v Sage nijak výjimečné. Výsledek funkce `mod` je prvek příslušné zbytkové třídy a dál se s ním tak pracuje.

Funkce mod samozřejmě je i v Matlabu, ale jak se dá čekat, výsledkem je číslo:

```
>> format longg
>> x=mod(20,12)
x =
     8
>> x^10
ans =
    1073741824
>> mod(ans,12)
ans =
     4
>>
```

Pokud tedy chceme v Matlabu provádět výpočty v rámci zbytkové třídy, musíme na výsledek operace opět aplikovat funkci mod. To ale někdy může být trochu problém, pokud je výsledek už příliš velký:

```
>> mod(2^55,12)
ans =
     8
>> mod(2^56,12)
ans =
     0
>>
```

Výsledek by měl být roven 4, ale 2^{56} už není v paměti uloženo přesně, takže operace mod s tak velkými a většími čísla dává nesprávné výsledky.

Můžeme si ale v Matlabu vytvořit vlastní funkce pro operace ve zbytkových třídách, které by dokázaly pracovat lépe. Se sčítáním asi nebude žádný problém a funkce se dá samozřejmě použít i na odečítání:

```
function c = plus_modulo(a,b,n)
%function c = plus_modulo(a,b,n)
%  scitani a+b modulo n

c=mod(a+b,n);
```



```
end
```

U násobení by mohl být výsledek překročit hranice přesnosti, proto funkci mod použijeme i na vstupní proměnné:

```
function c = krat_modulo(a,b,n)
%function c = krat_modulo(a,b,n)
%  nasobeni a*b modulo n

a1=mod(a,n);
b1=mod(b,n);
c=mod(a1*b1,n);

end
```

U mocniny využijeme algoritmus, který se běžně uvádí pro násobení, že totiž umocňujeme jenom na druhou a podle potřeby tuto mocninu násobíme s průběžným mezivýsledkem:

```
function c = mocnina_modulo(a,b,n)
%function c = mocnina_modulo(a,b,n)
%  mocnina a^b modulo n

if b<0, error('Zaporny exponent'); end
c=1;
z=mod(a,n);
while b>0
    if mod(b,2)
        c=mod(c*z,n);
    end
    b=floor(b/2);
    z=mod(z*z,n);
end

end
```

Tady u výpočtu druhé mocniny nemusíme mít strach, že výsledek by mohl být moc velký, protože funkci mod jsme použili v předchozím kroku. Funkce pak výpočty i s velkými exponenty zvládá bez problémů:

```
>> mocnina_modulo(2,56,12)
ans =
     4
>> mocnina_modulo(2,10000,12)
ans =
     4
>>
```

Aby náš výčet byl kompletní, vytvořme ještě funkci pro výpočet inverzního prvku modulo n . Při tom můžeme využít malou Fermatovu větu, která pro prvočíslo n dává

$$a^{n-1} \equiv 1 \pmod{n}$$

```
function c = inverze_modulo(a,n)
%function c = inverze_modulo(a,n)
%  inverzni privek 1/a modulo n
%  n musi byt prvocislo

if ~isprime(n)
    error('Modulo zaklad neni prvocislo');
end

c=mocnina_modulo(a,n-2,n);

end
```

Druhou možností je použít Eukleidův algoritmus pro dělení se zbytkem, který pro čísla a a b a jejich největší společný dělitel d dává čísla x , y , že platí

$$d = a \cdot x + b \cdot y.$$

Podle tohoto postupu můžeme inverzi a modulo n spočítat v případě, že a a n jsou nesoudělná. Při výpočtu můžeme využít funkci gcd, která může jako další výstupy vrátit zmiňované hodnoty x a y :

sage:

Kapitola 2

Algebraické výpočty

2.1 Zbytkové třídy

Sage umí pracovat se základními číselnými okruhy a tělesy:

```
sage: ZZ
Integer Ring
sage: QQ
Rational Field
sage: RR
Real Field with 53 bits of precision
sage: CC
Complex Field with 53 bits of precision
sage: Integers()
Integer Ring
sage: ZZ==Integers()
True
sage:
```

A ani vytvářet složitější struktury mu není cizí. O zbytkových třídách modulo n už tady byla řeč, podívejme se na ně trochu blíž:

```
sage: Z8=Integers(8)
sage: Z8
Ring of integers modulo 8
sage: Z8.list()
```

```

[0, 1, 2, 3, 4, 5, 6, 7]
sage: Z8.addition_table()
+ a b c d e f g h
+-----+
a| a b c d e f g h
b| b c d e f g h a
c| c d e f g h a b
d| d e f g h a b c
e| e f g h a b c d
f| f g h a b c d e
g| g h a b c d e f
h| h a b c d e f g

sage:

```

Vidíme, že prvky v tabulce sčítání jsou zobrazeny symbolicky, je samozřejmě možné způsob zobrazení nastavit. Taky si zobrazíme tabulku násobení

```

sage: Z8.addition_table(names='elements')
+ 0 1 2 3 4 5 6 7
+-----+
0| 0 1 2 3 4 5 6 7
1| 1 2 3 4 5 6 7 0
2| 2 3 4 5 6 7 0 1
3| 3 4 5 6 7 0 1 2
4| 4 5 6 7 0 1 2 3
5| 5 6 7 0 1 2 3 4
6| 6 7 0 1 2 3 4 5
7| 7 0 1 2 3 4 5 6

sage: Z8.multiplication_table(names='elements')
* 0 1 2 3 4 5 6 7
+-----+
0| 0 0 0 0 0 0 0 0
1| 0 1 2 3 4 5 6 7
2| 0 2 4 6 0 2 4 6
3| 0 3 6 1 4 7 2 5
4| 0 4 0 4 0 4 0 4

```

```
5| 0 5 2 7 4 1 6 3
6| 0 6 4 2 0 6 4 2
7| 0 7 6 5 4 3 2 1

sage:
```

Na jednotlivé prvky se můžeme dostat pomocí jejich indexu:

```
sage: x=Z8(3);x
3
sage: y=Z8(11);y
3
sage: x==y
True
sage:
```

Taky se lze dotazovat na různé vlastnosti jednotlivých prvků:

```
sage: x.is_zero()
False
sage: x.is_one()
False
sage: x.is_unit()
True
sage:
```

Všechny invertibilní prvky tedy zjistíme příkazem

```
sage: [[a,a.is_unit()] for a in Z8]
[[0, False],
 [1, True],
 [2, False],
 [3, True],
 [4, False],
 [5, True],
 [6, False],
 [7, True]]
sage: Z7=Integers(7)
```

```
sage: [[a,a.is_unit()] for a in Z7]
[[0, False],
 [1, True],
 [2, True],
 [3, True],
 [4, True],
 [5, True],
 [6, True]]
sage:
```

Okruh zbytkových tříd lze také vytvořit jako podíl $\mathbb{Z}/8\mathbb{Z}$:

```
sage: R8=ZZ.quotient(8)
sage: R8
Ring of integers modulo 8
sage: R8==Z8
True
sage:
```

Je také možné vytvořit aditivní grupu izomorfní zbytkové třídě modulo n vzhledem ke sčítání, jedná se ale o jiný objekt:

```
sage: R8=AdditiveAbelianGroup([8])
sage: R8
Additive abelian group isomorphic to Z/8
sage: R8.list()
[(0), (1), (2), (3), (4), (5), (6), (7)]
sage: R8==Z8
False
sage:
```

2.2 Grupy permutací

V Sage se grupy permutací nazývají symetrické grupy a k jejich vytváření se používá příkazem `SymmetricGroup`:

```
sage: S3=SymmetricGroup(3);S3
Symmetric group of order 3! as a permutation group
```



```

sage: S3.list()
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
sage: S4=SymmetricGroup(4)
sage: S4.list()
[(),
 (3,4),
 (2,3),
 (2,3,4),
 (2,4,3),
 (2,4),
 (1,2),
 (1,2)(3,4),
 (1,2,3),
 (1,2,3,4),
 (1,2,4,3),
 (1,2,4),
 (1,3,2),
 (1,3,4,2),
 (1,3),
 (1,3,4),
 (1,3)(2,4),
 (1,3,2,4),
 (1,4,3,2),
 (1,4,2),
 (1,4,3),
 (1,4),
 (1,4,2,3),
 (1,4)(2,3)]
sage:

```

Jednotlivé prvky (permutace) jsou tedy vyjádřeny pomocí cyklů. Pokud chceme vybrat nějaký prvek, odkážeme se na něj pomocí permutace, tedy pořadí čísel $1, \dots, n$:

```

sage: tau=S4([2,4,3,1]);tau
(1,2,4)
sage: tau.list()
[2, 4, 3, 1]

```

```

sage: [a.list() for a in S4]
[[1, 2, 3, 4],
 [1, 2, 4, 3],
 [1, 3, 2, 4],
 [1, 3, 4, 2],
 [1, 4, 2, 3],
 [1, 4, 3, 2],
 [2, 1, 3, 4],
 [2, 1, 4, 3],
 [2, 3, 1, 4],
 [2, 3, 4, 1],
 [2, 4, 1, 3],
 [2, 4, 3, 1],
 [3, 1, 2, 4],
 [3, 1, 4, 2],
 [3, 2, 1, 4],
 [3, 2, 4, 1],
 [3, 4, 1, 2],
 [3, 4, 2, 1],
 [4, 1, 2, 3],
 [4, 1, 3, 2],
 [4, 2, 1, 3],
 [4, 2, 3, 1],
 [4, 3, 1, 2],
 [4, 3, 2, 1]]
sage:

```

Tímto způsobem můžeme vyjádřit permutaci pomocí cyklů, případně naopak. Také můžeme permutace skládat a výsledek si vypisovat ve tvaru, který nám vyhovuje. Samozřejmě také můžeme určit inverzní permutaci:

```

sage: S4([3,2,4,1])
(1,3,4)
sage: S4((1,4,3))
(1,4,3)
sage: S4((1,4,3)).list()
[4, 2, 1, 3]
sage: p1=S4([2,4,1,3]);p1

```

```

(1,2,4,3)
sage: p2=S4([4,3,1,2]);p2
(1,4,2,3)
sage: p1*p2
(1,3,4)
sage: p2*p1
(1,3,2)
sage: p3=p2*p1;p3.list()
[3, 1, 2, 4]
sage: p1.inverse()
(1,3,4,2)
sage: p1.inverse().list()
[3, 1, 4, 2]
sage:

```

Na výpis tabulky operací pro grupu permutací slouží příkaz `cayley_table`, pokud jej ale použijeme bez dalších parametrů, je výsledek poněkud nepřehledný.

```

sage: S3.cayley_table()
* a b c d e f
+-----+
a| a b c d e f
b| b a d c f e
c| c e a f b d
d| d f b e a c
e| e c f a d b
f| f d e b c a

```

Proto si jednotlivé permutace označíme symboly, abychom je snáze identifikovali.

```

sage: S3.list()
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
sage:

```

Máme tam dva cykly, ty označíme `c1` a `c2`. Pak jsou tam tři permutace, které prohodí mezi sebou dva prvky, ty označíme postupně `r1`, `r3` a `r2` podle toho, který prvek zůstává na místě. No a označení `id` ja doufám jasné.

```

sage: S3.cayley_table(names=['id','r1','r3',
'c1','c2','r2'])
* id r1 r3 c1 c2 r2
+-----+
id| id r1 r3 c1 c2 r2
r1| r1 id c1 r3 r2 c2
r3| r3 c2 id r2 r1 c1
c1| c1 r2 r1 c2 id r3
c2| c2 r3 r2 id c1 r1
r2| r2 c1 c2 r1 r3 id

sage:

```

Z tabulky je například vidět, které prvky jsou inverzní ke kterým. Čtenář může popřemýšlet, jestli by se dalo podobným způsobem aspoň trochu zpřehlednit tabulku operací pro S_4 .

Poslední ukázkou z oblasti grup permutací budou kvaterniony. Jedná se o zobecnění komplexních čísel, kdy kromě imaginární jednotky i máme ještě dvě další j a k , přičemž platí

$$I^1 = j^2 = k^2 = -1, \quad ij = k, \quad jk = i, \quad ki = j.$$

Kvaterniony tvoří nekomutativní těleso, záměnou pořadí násobení u posledních tří rovností se otočí znaménko výsledku.

Uvedené imaginární jednotky spolu s ± 1 a svými opačnými hodnotami tvoří multiplikativní grupu řádu 8, která je isomorfní podgrupě grupy permutací 8 prvků:

```

sage: Q=QuaternionGroup();Q
Quaternion group of order 8 as a permutation group
sage: Q.list()
[(),
(1,2,3,4)(5,6,7,8),
(1,3)(2,4)(5,7)(6,8),
(1,4,3,2)(5,8,7,6),
(1,5,3,7)(2,8,4,6),
(1,6,3,8)(2,5,4,7),
(1,7,3,5)(2,6,4,8),

```

```
(1,8,3,6)(2,7,4,5]
sage: [a.list() for a in Q]
[[1, 2, 3, 4, 5, 6, 7, 8],
 [2, 3, 4, 1, 6, 7, 8, 5],
 [3, 4, 1, 2, 7, 8, 5, 6],
 [4, 1, 2, 3, 8, 5, 6, 7],
 [5, 8, 7, 6, 3, 2, 1, 4],
 [6, 5, 8, 7, 4, 3, 2, 1],
 [7, 6, 5, 8, 1, 4, 3, 2],
 [8, 7, 6, 5, 2, 1, 4, 3]]
sage:
```

Zobrazíme si tabulku operací, přičemž jednotlivé prvky označíme symboly uvedených imaginárních jednotek:

```
sage: Q.cayley_table(names=['1', 'i', '-1', '-i',
'j', '-k', '-j', 'k'])
*   1  i -1 -i  j -k -j  k
  +-----+
  1|  1  i -1 -i  j -k -j  k
  i|  i -1 -i  1  k  j -k -j
-1| -1 -i  1  i -j  k  j -k
-i| -i  1  i -1 -k -j  k  j
  j|  j -k -j  k -1 -i  1  i
-k| -k -j  k  j  i -1 -i  1
-j| -j  k  j -k  1  i -1 -i
  k|  k  j -k -j -i  1  i -1
sage:
```

Čtenář si může vyzkoušet, jestli přiřazení imaginárních jednotek jednotlivým permutacím je jednoznačně určeno, či zda snad jsou i jiné možnosti.

2.3 Polynomy

Také polynomy je možné v Sage definovat vícero způsoby.

```

sage: t=var('t')
sage: R = PolynomialRing(QQ, 't')
sage: S = QQ['t']
sage: S==R
True
sage: R2.<t>=QQ[]
sage: S2.<t>=PolynomialRing(QQ)
sage: R2==S2
True
sage: R2==R
True
sage:

```

S polynomy lze provádět různé operace:

```

sage: p1=(t+1)*(t+2);p1
t^2 + 3*t + 2
sage: p1^2
t^4 + 6*t^3 + 13*t^2 + 12*t + 4
sage: p1 in R
True
sage: p1.parent()
Univariate Polynomial Ring in t over Rational Field
sage: p1.is_irreducible()
False
sage: p1.factor()
(t + 1) * (t + 2)
sage: R.gen()
t
sage:

```

Dají se také vytvářet polynomy nad zbytkovými třídami modulo n :

```

sage: R8.<x>=Integers(8)[];R8
Univariate Polynomial Ring in x over
Ring of integers modulo 8
sage: p2=x^3+4*x-5

```

```

sage: p2
x^3 + 4*x + 3
sage: p2^2
x^6 + 6*x^3 + 1
sage: p2 in R8
True
sage: [p2(a) for a in Integers(8)]
[3, 0, 3, 2, 3, 4, 3, 6]
sage:

```

Polynomy více proměnných taky nejsou problémem:

```

sage: S2.<x,y>=PolynomialRing(ZZ);S2
Multivariate Polynomial Ring in x, y over Integer Ring
sage: f=(x^3+2*y^2*x)^2
sage: g=x^2*y^2
sage: d0=f.gcd(g);d0
x^2
sage:

```

Kapitola 3

Kombinatorické a grafové algoritmy

3.1 Kombinatorika

O kombinatorice jsem se už zmiňovali vícekrát, ukážeme si tedy ještě alespoň několik příkladů s použitím Sage.

Začneme karetním příkladem, který možná ocení hráči pokeru:

```
sage: Barvy=Set(["srdce","kary","piky","krize"])
sage: Hodnoty=Set([2,3,4,5,6,7,8,9,10,"kluk","dama",
"krak","eso"])
sage: Karty=CartesianProduct(Hodnoty,Barvy)
sage: C5=Combinations(Karty,5)
sage: C5.cardinality()
2598960
sage: binomial(Karty.cardinality(),5)
2598960
sage: C5.random_element()
[[4, 'krize'], [5, 'piky'], [5, 'krize'], [7, 'kary'],
[7, 'krize']]
sage:
```

Dostali jsme dvě pětky a dvě sedmičky, to není špatné. Ještě zahodíme tu čtverku a zkusíme full house:


```
sage: Karty.random_element()
[8, 'srdce']
sage:
```

Tak dnes to nevyšlo, snad příště.

Jako další příklad si zkusíme konstrukci Pascalova trojúhelníku:

```
sage: [[binomial(n,i) for i in range(n+1)] for n in
range(12)]
[[1],
 [1, 1],
 [1, 2, 1],
 [1, 3, 3, 1],
 [1, 4, 6, 4, 1],
 [1, 5, 10, 10, 5, 1],
 [1, 6, 15, 20, 15, 6, 1],
 [1, 7, 21, 35, 35, 21, 7, 1],
 [1, 8, 28, 56, 70, 56, 28, 8, 1],
 [1, 9, 36, 84, 126, 126, 84, 36, 9, 1],
 [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1],
 [1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]]
sage:
```

Nesmíme při tom zapomenout, že `range(n)` dává hodnoty od 0 do $n - 1$.

Další kombinatorickou úlohou je rozklad přirozeného čísla na součty:

```
sage: Q=Partitions(6)
sage: Q
Partitions of the integer 6
sage: Q.cardinality()
11
sage: Q.list()
[[6],
 [5, 1],
 [4, 2],
 [4, 1, 1],
 [3, 3],
```

```
[3, 2, 1],
[3, 1, 1, 1],
[2, 2, 2],
[2, 2, 1, 1],
[2, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1]]
sage:
```

Podobně funguje funkce `Compositions`, u níž se ale bere v potaz i pořadí:

```
sage: Q1=Compositions(5);Q1
Compositions of 5
sage: Q1.cardinality()
16
sage: Q1.list()
[[1, 1, 1, 1, 1],
 [1, 1, 1, 2],
 [1, 1, 2, 1],
 [1, 1, 3],
 [1, 2, 1, 1],
 [1, 2, 2],
 [1, 3, 1],
 [1, 4],
 [2, 1, 1, 1],
 [2, 1, 2],
 [2, 2, 1],
 [2, 3],
 [3, 1, 1],
 [3, 2],
 [4, 1],
 [5]]
sage:
```

Dalším známým kombinatorickým problémem je nalezení všech možných přípustných uzávorkování n páry závorek, tedy laicky řečeno, pravé závorky nesmí převažovat nad levými. Počet všech možností dávají Catalanova čísla:

```
sage: for i in range(11): print catalan_number(i)
1
```

```
1
2
5
14
42
132
429
1430
4862
16796
sage:
```

Sage ale umí také najít všechna přípustná uzávorkování:

```
sage: D=DyckWords(3);D
Dyck words with 3 opening parentheses and 3 closing parentheses
sage: D.cardinality()
5
sage: D.list()
[[1, 0, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 0],
 [1, 1, 0, 0, 1, 0],
 [1, 1, 0, 1, 0, 0],
 [1, 1, 1, 0, 0, 0]]
sage:
```

Lepší možná bude zobrazení skutečně pomocí závorek:

```
sage: for i in D: print i
()()()
()(())
(())()
(()())
((()))
sage: D2=DyckWords(4)
sage: D2.cardinality()
14
sage: for i in D2: print i
```

```
()()()()
()()()()
()()()()
()()()()
()((()()))
()()()()
()()()()
()()()()
()()()()
()()()()
()()()()
((()()))()
((()()))()
((()()))()
(((())))
```

sage:

Jako poslední příklad si ukážeme práci s abecedou:

```
sage: W=Words("xy");W
Words over {'x', 'y'}
sage: W.cardinality()
+Infinity
sage: i1=iter(W)
sage: for k in range(20): print i1.next()

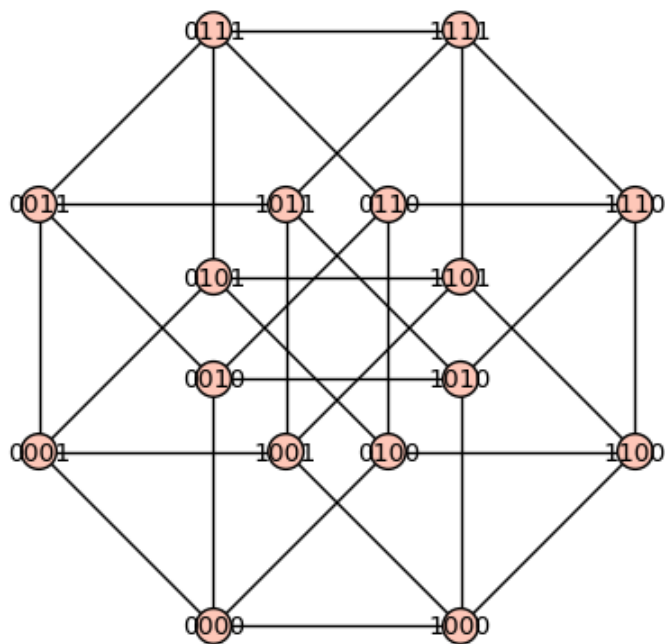
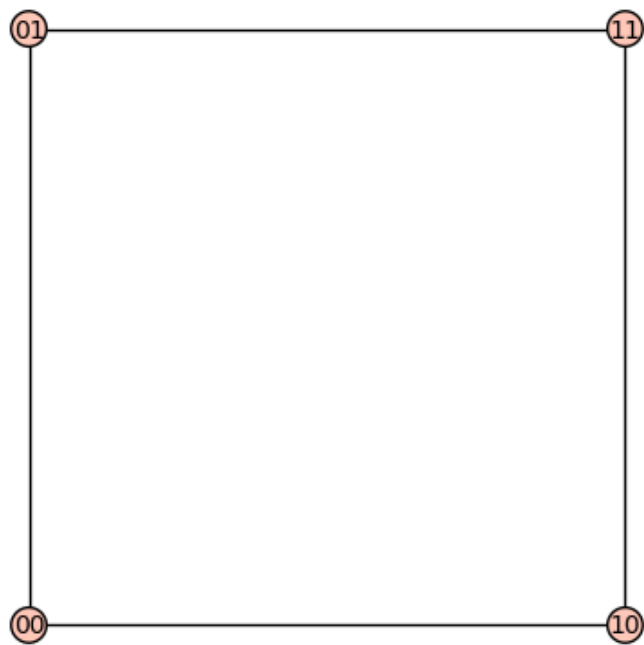
x
y
xx
xy
yx
yy
xxx
xxy
xyx
xyy
yxx
yxy
yyx
```

```
yyy
xxxx
xxxxy
xxyx
xxyy
xyxx
sage:
```

3.2 Grafy

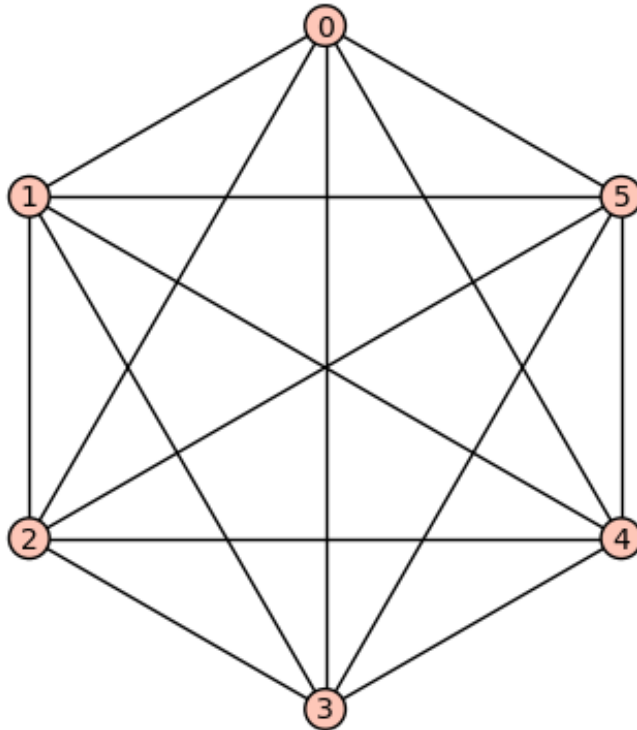
Ukážeme si, jak v Sage generovat některé základní grafy. Jako první to bude hyperkrychle v n dimenzích. Pro $n = 2$ dostaneme samozřejmě čtverec, pro $n = 4$ je to poněkud zajímavější.

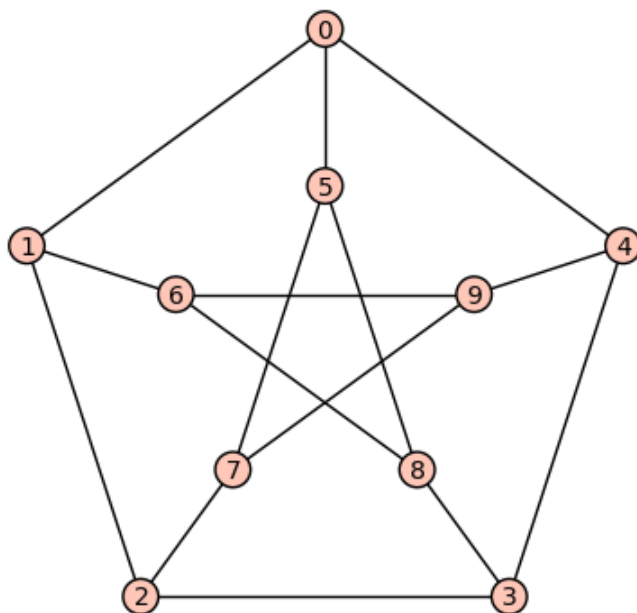
```
sage: C = graphs.CubeGraph(2);C
2-Cube: Graph on 4 vertices
sage: C.show()
sage: C = graphs.CubeGraph(4);C.show()
sage:
```



Důležitý je samozřejmě úplný graf či Petersenův graf:

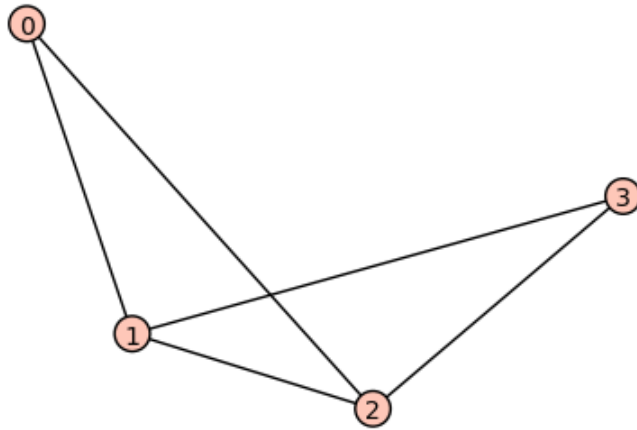
```
sage: graphs.CompleteGraph(6).show()  
graphs.PetersenGraph().show()  
sage:
```





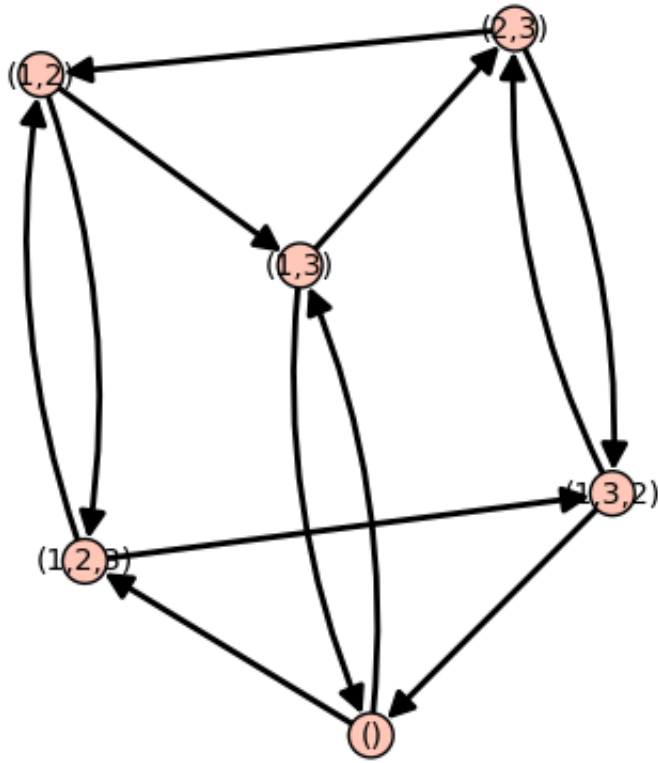
Grafově lze také reprezentovat matice, kdy 1 nebo 0 říkájí v matici, zda uzly očíslované indexy řádků resp. sloupců, jsou nebo nejsou spojeny hranou. Přitom na hlavní diagonále musejí být nuly. Příslušné grafy často mohou pomoci například při faktorizaci matice.

```
sage: M1=Matrix([[0,1,1,0],[1,0,1,1],[1,1,0,1],[0,1,1,0]])
sage: M1
[0 1 1 0]
[1 0 1 1]
[1 1 0 1]
[0 1 1 0]
sage: G1=Graph(M1);G1.show()
sage:
```

A nesmíme zapomínat na orientované grafy, jako je třeba Cayleyho graf dané grupy:

```
sage: G = DihedralGroup(3)
sage: G
Dihedral group of order 6 as a permutation group
sage: G.list()
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
sage: G.cayley_graph()
Digraph on 6 vertices
sage: G.cayley_graph().show()
sage:
```



Literatura

- [1] Fuchs, E.: *Kombinatorika a teorie grafů*. SPN, 1986.
- [2] Kuroš, A. G. c.: *Kapitoly z obecné algebry*. Academia, 1977.