480 - April 30 2008 -- combinatorics, part 2

Open Source Quote of the Day

> I've always been saddened that my University would use FOSS software but would not give us a single class about what it was all about. In some ways this was a breach of the moral contract you take when using free or open-source software. A bit like those gazillion companies that use FOSS and believe the only thing they should know it's free like as in free beer. This hurts everyone who's passionate enough about FOSS I think.
>
> -- Sylvain Hellegouarch (Mon, Apr 28 2008, Google Summer of Code mentor)

Combinatorics, part 2

Combinatorial Structures

Sage supports dozens of combinatorial ways to form sets of objects from other sets. These are incredibly useful in enumeration and counting problems. The functionality described below in Sage is due almost entirely to Mike Hansen, and is just a small sampling of the range of "combinatorial classes" available in Sage.

Cartesian Products

```
C = CartesianProduct(prime_range(10), prime_range(10,20),
['a','b','c'])
C
```

```
    Cartesian product of [2, 3, 5, 7], [11, 13, 17, 19], ['a', 'b', 'c
```

```
C.list()
```

```
    [[2, 11, 'a'], [2, 11, 'b'], [2, 11, 'c'], [2, 13, 'a'], [2, 13,
    'b'], [2, 13, 'c'], [2, 17, 'a'], [2, 17, 'b'], [2, 17, 'c'], [2,
    19, 'a'], [2, 19, 'b'], [2, 19, 'c'], [3, 11, 'a'], [3, 11, 'b'],
    [3, 11, 'c'], [3, 13, 'a'], [3, 13, 'b'], [3, 13, 'c'], [3, 17,
    'a'], [3, 17, 'b'], [3, 17, 'c'], [3, 19, 'a'], [3, 19, 'b'], [3,
    19, 'c'], [5, 11, 'a'], [5, 11, 'b'], [5, 11, 'c'], [5, 13, 'a'],
    [5, 13, 'b'], [5, 13, 'c'], [5, 17, 'a'], [5, 17, 'b'], [5, 17,
    'c'], [5, 19, 'a'], [5, 19, 'b'], [5, 19, 'c'], [7, 11, 'a'], [7,
    11, 'b'], [7, 11, 'c'], [7, 13, 'a'], [7, 13, 'b'], [7, 13, 'c'],
    [7, 17, 'a'], [7, 17, 'b'], [7, 17, 'c'], [7, 19, 'a'], [7, 19,
    'b'], [7, 19, 'c']]
```

```
C.count()
```

```
    48
```

```
C = CartesianProduct(['Jim', 'Dwight', 'Michael'],['Pam',
'Angela'])
```

```
C
```

```
    Cartesian product of ['Jim', 'Dwight', 'Michael'], ['Pam', 'Angela
```

```
for X in C:
  print '%s and %s'%tuple(X)
```

```
    Jim and Pam
    Jim and Angela
    Dwight and Pam
    Dwight and Angela
    Michael and Pam
    Michael and Angela
```

```
C.random()
```

```
    ['Jim', 'Angela']
```

```

```

Combinations

```
C = Combinations(['Jim', 'Pam', 'Dwight', 'Angela', 'Michael'], 2)
```

```
C
```

```
Combinations of ['Jim', 'Pam', 'Dwight', 'Angela', 'Michael'] of
length 2
```

```
len(C)
```

```
10
```

```
C.list()
```

```
[['Jim', 'Pam'], ['Jim', 'Dwight'], ['Jim', 'Angela'], ['Jim',
'Michael'], ['Pam', 'Dwight'], ['Pam', 'Angela'], ['Pam',
'Michael'], ['Dwight', 'Angela'], ['Dwight', 'Michael'], ['Angela'
'Michael']]
```

```
for X in C.list():
   print '%s and %s'%tuple(X)
```

```
Jim and Pam
Jim and Dwight
Jim and Angela
Jim and Michael
Pam and Dwight
Pam and Angela
Pam and Michael
Dwight and Angela
Dwight and Michael
Angela and Michael
```

Compositions

Compositions: Ordered lists of positive integers that sum to 4.

```
C = Compositions(4)
C
```

```
Compositions of 4
```

```
for v in C: print v
```

```
[1, 1, 1, 1]
[1, 1, 2]
[1, 2, 1]
[1, 3]
[2, 1, 1]
[2, 2]
[3, 1]
[4]
```

```
len(C)
```

```
8
```

```
C.count()
```

```
8
```

Partitions

```
P = Partitions(5)
P
```

```
Partitions of the integer 5
```

```
len(P)
```

```
7
```

```
P.list()
```

```
[[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1], [1, 1, 1
1, 1]]
```

```
for p in P:
   print p
   print ferrers_diagram(p)
   print
```

```
[5]
*****

[4, 1]
****
*

[3, 2]
***
**

[3, 1, 1]
***
*
*

[2, 2, 1]
**
**
*

[2, 1, 1, 1]
**
*
*
*

[1, 1, 1, 1, 1]
*
*
*
*
*
```

```
len(Partitions(100))
```

```
190569292
```

```
Partitions(10^6).count()
```

```
1471684986358223398631004760609895943484030484439142125334612747356611741891861827633014887398359755584201537413060028809592938734712322703278495780019327843960720642286590487130201709718407610256769860846908142829356706929785991290519899445490672219997823452874987402228822985013676756629478188749468787900382469998819772920063206687359966622738167982662134824172084466310274280019181321981771806511234542595026728424452592296781193448139994664730105742564359159498918148528535137055139947671998169145902201559910195960141747407154307500221848958152093390124817344694483193232801506653840429944179587751761294916248142479998802936507195257074485047571662771760339144249511382329819526300833648982604583771220245530499638214460285318320045190465919683027875374181184860006120168525935427419805046267245473237321845833427512524227465399130174076941280847400834221799928607110833630331629828910244464969680539541679187548001086367740220231284676469197750223485625207477418433436578015341307041975530375169707999287040285677841619347472368171772154046664303121563000346710467381 8
```

Set Partitions

```
S = SetPartitions(['Jim', 'Pam', 'Dwight'])
S
```

```
Set partitions of ['Jim', 'Pam', 'Dwight']
```

```
for X in S.list():
   print X
```

```
{{'Jim', 'Pam', 'Dwight'}}
{{'Jim'}, {'Pam', 'Dwight'}}
{{'Pam'}, {'Jim', 'Dwight'}}
{{'Dwight'}, {'Jim', 'Pam'}}
{{'Pam'}, {'Jim'}, {'Dwight'}}
```

Ordered Set Partitions

An ordered set partition p of a set s is a partition of s, into subsets called parts and represented as a list of sets. By extension, an ordered set partition of a nonnegative integer n is the set partition of the integers from 1 to n. The number of ordered set partitions of n is called the n-th ordered Bell number.

```
for X in OrderedSetPartitions(3):
  print X
```

```
[{1}, {2}, {3}]
[{1}, {3}, {2}]
[{2}, {1}, {3}]
```

```
[{3}, {1}, {2}]
[{2}, {3}, {1}]
[{3}, {2}, {1}]
[{1}, {2, 3}]
[{2}, {1, 3}]
[{3}, {1, 2}]
[{1, 2}, {3}]
[{1, 3}, {2}]
[{2, 3}, {1}]
[{1, 2, 3}]
```

```
P = OrderedSetPartitions(4, [1,2,1])
P
```

```
Ordered set partitions of {1, 2, 3, 4} into parts of size [1, 2, 1
```

```
P.count()
```

```
12
```

```
for X in P: print X
```

```
[{1}, {2, 3}, {4}]
[{1}, {2, 4}, {3}]
[{1}, {3, 4}, {2}]
[{2}, {1, 3}, {4}]
[{2}, {1, 4}, {3}]
[{3}, {1, 2}, {4}]
[{4}, {1, 2}, {3}]
[{3}, {1, 4}, {2}]
[{4}, {1, 3}, {2}]
[{2}, {3, 4}, {1}]
[{3}, {2, 4}, {1}]
[{4}, {2, 3}, {1}]
```

```

```

```

```

Permutations

```
p = Permutations(['Jim', 'Pam', 'Dwight'])
p
```

```
Permutations of the set ['Jim', 'Pam', 'Dwight']
```

```
for z in p:
  print z
```

```
['Jim', 'Pam', 'Dwight']
['Jim', 'Dwight', 'Pam']
['Pam', 'Jim', 'Dwight']
['Pam', 'Dwight', 'Jim']
```

```
    ['Dwight', 'Jim', 'Pam']
    ['Dwight', 'Pam', 'Jim']
```

```
len(p)
```

```
    6
```

Subsets

```
S = Subsets(['Jim', 'Pam', 'Dwight'])
S
```

```
    Subsets of {'Jim', 'Pam', 'Dwight'}
```

```
for X in S.list():
   print X
```

```
    {}
    {'Jim'}
    {'Pam'}
    {'Dwight'}
    {'Jim', 'Pam'}
    {'Jim', 'Dwight'}
    {'Pam', 'Dwight'}
    {'Jim', 'Pam', 'Dwight'}
```

```
S.count()
```

```
    8
```

Dyck Words

Dyck words are the expressions containing n pairs of parentheses that are correctly matched.

```
D = DyckWords(3)
```

```
for x in D:
  print x
```

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

```
d = D[0]
```

There are many combinatorial structures that are in bijection with Dyck words. This is the *stuff* of combinatorics...

```
d.to_tableau()
```

```
    [[4, 5, 6], [1, 2, 3]]
```

```
d.to_ordered_tree()
```

```
    Traceback (click to the left for traceback)
    ...
    NotImplementedError: TODO
```

```
d.to_tableau()
```

```
    [[4, 5, 6], [1, 2, 3]]
```

```
d.to_noncrossing_partition()
```

```
    [[1, 2, 3]]
```

```
d.a_statistic()
```

```
    3
```

```

```

```
@interact
def _(n=(2..8)):
   D = DyckWords(n)
   print "The %s %s."%(D.count(), D)
   for X in D:
       print X
```

n

```
The 2 Dyck words with 2 opening parentheses and 2 closing
parentheses.
(())
()()
```

Graph Paths

Another combinatorial class in Sage is the collection of paths in a directed graph.

```
G = DiGraph({1:[2,2,3], 2:[3,4], 3:[4], 4:[5,5]}, multiedges=True)
G.plot3d(spin=True)
```

```
p = GraphPaths(G); p
```

```
    Paths in Multi-digraph on 5 vertices
```

```
len(p)
```

```
    37
```

```
p.random()
```

```
    [2, 4, 5]
```

```
list(p)
```

```
    [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5], [1, 2, 3,
    5], [1, 2, 4], [1, 2, 4, 5], [1, 2, 4, 5], [1, 2], [1, 2, 3], [1,
    3, 4], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 4], [1, 2, 4, 5],
    [1, 2, 4, 5], [1, 3], [1, 3, 4], [1, 3, 4, 5], [1, 3, 4, 5], [2],
    [2, 3], [2, 3, 4], [2, 3, 4, 5], [2, 3, 4, 5], [2, 4], [2, 4, 5],
    [2, 4, 5], [3], [3, 4], [3, 4, 5], [3, 4, 5], [4], [4, 5], [4, 5],
    [5]]
```