glynnspiespermanent

```
def genmatrix(d):
    mat = []
    for x in xrange(d):
        row = []
        for y in xrange(d):
            row.append([0,1][randrange(0,2)])
        mat.append(row)
    return matrix(mat)
```

```
from time import time
```

```
M = genmatrix(20)
```

```
print M.str()
```

```
[1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 1]
[0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1]
[0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0]
[1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 1]
[0 0 1 1 0 1 1 1 0 1 0 1 0 0 0 1 1 0 0 0]
[0 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1]
[0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1]
[1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 1 0 1]
[1 1 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1]
[0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0]
[0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 0 1 1 0]
[0 0 0 1 1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0]
[1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 1 1 0]
[1 0 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0]
[0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 1]
[0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 0]
[0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0]
[1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1]
[1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1]
[1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0]
```

```
t = time()
print M.permanent()
print time()-t
```

```
446579339853
2.53839111328
```

```
t = time()
print M.permanent(algorithm="ButeraPernici")
print time()-t
```

```
446579339853
6.77643299103
```

```
def perm_glynn(M):
    col_sums = [sum(c) for c in zip(*M)]
    n = M.dimensions()[0]
    total = 0
    old_gray = 0
    d_i = +1
    powers_of_2_dict = {2**i:i for i in range(n)}
    num_loops = 2**(n-1)

    for num in xrange(1, num_loops + 1):
        prod = 1
        for i in col_sums:
            prod *= i
        total += d_i * prod

        new_gray = num ^^ (num >> 1)
        diff = old_gray ^^ new_gray
        diff_index = powers_of_2_dict[diff]
        new_vector = M.row(diff_index)
        dir = 2 * cmp(old_gray,new_gray)

        for i in range(n):
            col_sums[i] += new_vector[i] * dir

        d_i = -d_i
        old_gray = new_gray

    return total / num_loops
```

```
t=time()
print perm_glynn(M)
print time()-t
```

```
446579339853
5.79403400421
```

```
def perm_spies(M):


    row_sums = [sum(r) for r in M]
    n = M.dimensions()[0]
    total = 0
    old_gray = 0
    x_i = +1
    powers_of_2_dict = {2**i:i for i in range(n)}
    num_loops = 2**(n-1)
```

```
    for num in xrange(1, num_loops + 1):
        prod = 1
        for x in row_sums:
            prod *= x
        total += x_i * prod
        new_gray = num ^^ (num >> 1)
        diff = old_gray ^^ new_gray
        diff_index = powers_of_2_dict[diff]
        new_vector = M.column(diff_index)
        dir = 2 * cmp(old_gray,new_gray)

        for i in range(n):
            row_sums[i] += new_vector[i] * dir

        x_i = -x_i
        old_gray = new_gray

    return total / num_loops
```

```
t=time()
print perm_spies(M)
print time()-t
```

```
446579339853
5.71181416512
```

```
M.permanent?
```

File: /home/jaap/sage/SageMath/src/sage/matrix/matrix2.pyx

Type: <type 'builtin_function_or_method'>

Definition: M.permanent(algorithm='Ryser')

Docstring:

Return the permanent of this matrix.

Let $A = (a_{i,j})$ be an $m \times n$ matrix over any commutative ring with $m \leq n$. The permanent of A is

$$\mathrm{per}(A) = \sum_{\pi} a_{1,\pi(1)} a_{2,\pi(2)} \cdots a_{m,\pi(m)}$$

where the summation extends over all one-to-one functions π from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$.

The product $a_{1,\pi(1)} a_{2,\pi(2)} \cdots a_{m,\pi(m)}$ is called *diagonal product*. So the permanent of an $m \times n$ matrix A is the sum of all the diagonal products of A.

By default, this method uses Ryser's algorithm, but setting `algorithm` to "ButeraPernici" you can use the algorithm of Butera and Pernici (which is well suited for band matrices, i.e. matrices whose entries are concentrated near the diagonal).

INPUT:

- `A` – matrix of size $m \times n$ with $m \leq n$
- `algorithm` – either "Ryser" (default) or "ButeraPernici". The Butera-Pernici algorithm takes advantage of presence of zeros and is very well suited for sparse matrices.

ALGORITHM:

The Ryser algorithm is implemented in the method `_permanent_ryser()`. It is a modification of theorem 7.1.1. from Brualdi and Ryser: Combinatorial Matrix Theory. Instead of deleting columns from A, we choose columns from A and calculate the product of the row sums of the selected submatrix.

The Butera-Pernici algorithm is implemented in the function `permanental_minor_polynomial()`. It takes advantage of cancellations that may occur in the computations.

EXAMPLES:

```
sage: A = ones_matrix(4,4)
sage: A.permanent()
24

sage: A = matrix(3,6,[1,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,1,1])
sage: A.permanent()
36
sage: B = A.change_ring(RR)
sage: B.permanent()
36.0000000000000
```

The permanent above is directed to the Sloane's sequence OEIS sequence A079908 ("The Dancing School Problems") for which the third term is 36:

```
sage: oeis(79908)                          # optional -- internet
A079908: Solution to the Dancing School Problem with 3 girls and n+3 boys: f(3,n).
sage: _(3)                                 # optional -- internet
36

sage: A = matrix(4,5,[1,1,0,1,1,0,1,1,1,1,1,0,1,0,1,1,1,0,1,0])
sage: A.permanent()
32
```

A huge permanent that can not be reasonably computed with the Ryser algorithm (a 50×50 band matrix with width 5):

```
sage: n, w = 50, 5
sage: A = matrix(ZZ, n, n, lambda i,j: (i+j)%5 + 1 if abs(i-j) <= w else 0)
sage: A.permanent(algorithm="ButeraPernici")
577669727355110970369624817108922684046701056046769329 08
```

See Minc: Permanents, Example 2.1, p. 5.

```
sage: A = matrix(QQ,2,2,[1/5,2/7,3/2,4/5])
sage: A.permanent()
103/175

sage: R.<a> = PolynomialRing(ZZ)
sage: A = matrix(R,2,2,[a,1,a,a+1])
sage: A.permanent()
a^2 + 2*a

sage: R.<x,y> = PolynomialRing(ZZ,2)
sage: A = matrix(R,2,2,[x, y, x^2, y^2])
sage: A.permanent()
x^2*y + x*y^2
```

AUTHORS:

- Jaap Spies (2006-02-16 and 2006-02-21)

```
R.<a,b,c,d,e,f,g,h,i> = PolynomialRing(ZZ)
```

```
A = matrix(R,3,3,[a,b,c,d,e,f,g,h,i])
```

```
perm_glynn(A)
```

```
    c*e*g + b*f*g + c*d*h + a*f*h + b*d*i + a*e*i
```

```

```