

MLC Lab Visit - More Combinatorics

Mth 355 Fall 2001 Assignment 5 (at end). Due Oct 31.

Mth 355 (a.k.a. Mth 399) Oct 23 2001 Maple 6

Bent E. Petersen

Filename: 355f2001-assign-05-combinat.mws

Maple has quite a bit of combinatorial support available in the `combstruct` and `combinat` packages. This Worksheet is a summary of some of the combinatorial computations you can do with Maple. At the end of the worksheet are a few problems which constitute Assignment 5.

```
[ > restart;
```

```
[ > with(combstruct);
```

```
      [allstructs, count, draw, finished, gfeqns, gfsolve, iterstructs, nextstruct]
```

```
[ > with(combinat);
```

```
Warning, the protected name Chi has been redefined and unprotected
```

```
[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart, encodepart, fibonacci, firstpart, graycode, inttovec, lastpart, multinomial, nextpart, numbc comb, numbc comp, numbp art, numbp erm, partition, permute, powerset, prevpart, randcomb, randpart, randperm, stirling1, stirling2, subsets, vectoint]
```

Permutations

The `allstructs()` function produces a list of all permutations of the specified objects. The function `iterstructs()` allows one to iterate over the list of objects which would be returned by `allstructs()`, without first returning the list.

Here are the 2-permutations of `[a,b,c,d]`

```
[ > allstructs(Permutation([a,b,c,d]), size=2);
```

```
      [[a, b], [a, c], [a, d], [b, a], [b, c], [b, d], [c, a], [c, b], [c, d], [d, a], [d, b], [d, c]]
```

If the argument provided is an integer `n`, rather than a list, it is treated as the list `[1,...,n]`. Thus

```
[ > allstructs(Permutation(4), size=2);
```

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

To use an iterator instead is simple

```
[ > iter1:=iterstructs(Permutation([a,b,c,d]), size=2):
```

```
> while not finished(iter1) do
>   nextstruct(iter1);
> od;
```

```
[a, b]
[b, a]
[a, c]
[c, a]
[a, d]
[d, a]
[b, c]
[c, b]
[b, d]
[d, b]
[c, d]
[d, c]
```

Of course you will want to do something useful inside the do-loop.

You can also do permutations with repeated objects.

```
> allstructs(Permutation([a,b,c,c]), size=2);
[[a, b], [a, c], [b, a], [b, c], [c, a], [c, b], [c, c]]
```

A special keyword, `allsizes`, is used if you want all r -permutations for each size of r .

```
> allstructs(Permutation([a,b,c]), size=allsizes);
[[ ], [a], [b], [c], [a, b], [a, c], [b, a], [b, c], [c, a], [c, b], [a, b, c], [a, c, b], [b, a, c],
 [b, c, a], [c, a, b], [c, b, a]]
```

Note the 0-permutation, or empty permutation, is also returned. Be careful about that.

The `count()` function returns a count of the number of structures of the given type, for example permutations.

```
> count(Permutation([a,b,c]), size=allsizes);
16
> count(Permutation([a,b,c]), size=3);
6
```

The `draw()` function returns a randomly selected structure of the type specified, for example, a

permutation.

```
> draw(Permutation([a,b,c,d]), size=3);  
[d, a, c]
```

You can also use `randperm()` to obtain a random permutation.

```
> randperm([a,b,c,d]);  
[a, d, b, c]
```

Note you can not specify the size in `randperm()`.

You can also use the `permute()` function to obtain permutations.

```
> permute([a,b,c,d], 2);  
[[a, b], [a, c], [a, d], [b, a], [b, c], [b, d], [c, a], [c, b], [c, d], [d, a], [d, b], [d, c]]  
> permute([a,b,c]);  
[[a, b, c], [a, c, b], [b, a, c], [b, c, a], [c, a, b], [c, b, a]]  
> permute([a,a,b], 2);  
[[a, a], [a, b], [b, a]]
```

Combinations

Combinations work much the same as permutations.

```
> allstructs(Combination([a,b,c,d]), size=2);  
[[a, b], [a, c], [a, d], [b, c], [b, d], [c, d]]
```

If the argument provided is an integer `n`, rather than a list, it is treated as the set $\{1, \dots, n\}$. Thus

```
> allstructs(Combination(4), size=2);  
{ {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4} }
```

Iterators works as they do for permutations.

```
> iter2:=iterstructs(Combination([a,b,c,d]), size=2):  
> while not finished(iter2) do  
>   nextstruct(iter2);  
> od;  
[a, b]  
[a, c]
```

```
[a, d]
[b, c]
[b, d]
[c, d]
```

We can consider lists with repeated elements.

```
> allstructs(Combination([a,b,c,c],size=2);
[[b, c], [a, b], [a, c], [c, c]]
```

Note however, for sets repeated elements do not make sense. If your data is presented as a set multiple occurrences of elements will be treated as single occurrences (as is correct). Be careful.

```
> allstructs(Combination({a,b,c,c},size=2);
{{a, b}, {a, c}, {b, c}}
```

Again, allsizes is available.

```
> allstructs(Combination([a,b,c,c],size=allsizes);
[[ ], [c], [c, c], [b], [b, c], [b, c, c], [a], [a, c], [a, c, c], [a, b], [a, b, c], [a, b, c, c]]
```

or

```
> allstructs(Combination({a,b,c,c},size=allsizes);
{{ }, {a, b}, {a, c}, {b, c}, {a, b, c}, {c}, {a}, {b}}
```

Again note the difference between lists and sets. The set {a,b,c,c} is really the same as the set {a,b,c}.

We can obtain random combinations in two different ways (depending on the packages loaded).

```
> draw(Combination(6),size=3);
{2, 3, 4}
> randcomb(6,3);
{1, 2, 3}
```

We can count without listing as well:

```
> count(Combination(9),size=4);
126
```

or

```
> numbcomb(9,4);
126
> numbcomb([a,a,b,b,b]);
```

12

```
> numbcomb([a, a, b, b, b], 2);
```

3

Generalized Permutations and Combinations

We can select r objects, without regard to order, from a set of n objects (distinguishable since elements of a set) in $C(n,r) = P(n,r)/r! = n!/(r!(n-r)!)$ ways, because $P(n,r)$ is the number of ways to choose the r objects in order, and if we disregard the order, we clearly have to divide by $r!$. $C(n,r)$ is called $\text{binomial}(n,r)$ in Maple. It is the number of combinations of n distinct objects taken r at a time.

Let X be a set of cardinality n where $n = n_1 + n_2 + \dots + n_m$ and let $f: X \rightarrow \{1, 2, \dots, m\}$. Then f determines an ordered partition of X (that is, a list of nonempty subsets with union X). Suppose the i -th subset, the preimage of i , contains n_i points. How many such functions are there? Well, we can select $C(n, n_1)$ elements to map to 1, then $C(n - n_1, n_2)$ elements to map to 2, and so forth. Thus the number of functions with the preimage of i having cardinality n_i , $i = 1, \dots, m$ is $C(n, n_1) * C(n - n_1, n_2) * \dots * C(n - n_1 - \dots - n_{m-1}, n_m) = P(n; n_1, n_2, \dots, n_m)$; that is, $\text{multinomial}(n, n_1, \dots, n_m)$.

Thus, given n distinct objects and m numbered containers, there are $\text{multinomial}(n, n_1, \dots, n_m)$ ways of placing the objects in the containers, with n_k objects in the k -th container (but with no particular order within the containers). Here we are essentially dealing with m combinations.

Here's another way to look at it: Given n objects where there are n_k objects of type (color, etc.) k , $k=1, \dots, m$, there are $\text{multinomial}(n, n_1, \dots, n_m)$ ways to arrange them in a row (argue by choosing the location). Here we care about the order - essentially we are dealing with permutations with replacement.

Here's an example - the number of strings of length 12 which contain 4 a's, 3 b's, 2 c's and 3 d's is

```
> multinomial(12, 4, 3, 2, 3);
```

277200

```
> count(Permutation([a, a, a, a, b, b, b, c, c, d, d, d]), size=12);
```

277200

```
> numbperm([a, a, a, a, b, b, b, c, c, d, d, d], 12);
```

277200

Note that $\text{binomial}(n, k) = \text{multinomial}(n, k, n-k)$

Allocation of Identical Objects

Assume we wish to allocate m identical objects to n distinct cells. We can think of the cells as being ordered in a line and separated by $n-1$ walls. Then we have $m+n-1$ locations into which to place the walls and the objects. But the objects go into the remaining places after we have placed the walls. Thus $\text{binomial}(m+n-1, n-1) = \text{binomial}(m+n-1, m)$ is the number of ways we can allocate the objects to the cells.

If we have to place at least one object within each cell then n objects are consumed and we need only consider placing $m-n$ objects. Replacing m by $m-n$ above we see there are $\text{binomial}(m-1, n-1) = \text{binomial}(m-1, m-n)$ ways.

We can think of each cell as being chosen as many times as the number of objects in it. Then we see that the number of ways to choose m objects with repetition from a set of n distinct objects is $\text{binomial}(m+n-1, n-1) = \text{binomial}(m+n-1, m)$.

```
> allstructs(Combination([a,a,a,b,b,b,c,c,c,d,d,d]),size=2);
      [[c,d],[b,d],[b,c],[a,b],[a,c],[a,d],[c,c],[a,a],[b,b],[d,d]]
> count(Combination([a,a,a,b,b,b,c,c,c,d,d,d]),size=2);
      10
> binomial(4+2-1,2); binomial(4+2-1,3);
      10
      10
```

In the first two commands we are choosing 2 objects, with repetition, from {a,b,c,d}. In the last two commands we are allocating 4 identical objects to 2 distinct cells.

Partitions and Compositions of a natural number

```
> allstructs(Partition(6),size=3);
      [[1,1,4],[1,2,3],[2,2,2]]
```

We can also use the `partition()` function provided by the `combinat` package.

```
> partition(6);
[[1,1,1,1,1,1],[1,1,1,1,2],[1,1,2,2],[2,2,2],[1,1,1,3],[1,2,3],[3,3],[1,1,4],
 [2,4],[1,5],[6]]
```

If we just want the partitions of length 3 as above then `partition(6)` makes us work a bit harder than `allstructs(Partition(6),size=3)` does. Here is one way to do it.

```

> for prt in partition(6) do if nops(prt)=3 then print(prt); fi; od;
      [2, 2, 2]
      [1, 2, 3]
      [1, 1, 4]

```

Compositions of a number may be easily computed:

```

> allstructs(Composition(6), size=3);
[
  [1, 2, 3], [1, 1, 4], [2, 2, 2], [2, 3, 1], [1, 4, 1], [3, 2, 1], [4, 1, 1], [1, 3, 2], [3, 1, 2], [2, 1, 3]
]

```

If we just want the number of partitions or compositions we use the `count()` function just as we did for permutations.

```

> count(Partition(6), size=3);
      3
> count(Composition(6), size=3);
      10

```

We can also use the functions `numbpart()` and `numbcomp()` from the `combinat` package:

```

> numbpart(6);
      11
> numbcomp(6, 3);
      10

```

Be careful - `numbcomp(n)` is not supported and yields an error message and `numbpart(n,m)` is also not supported, but yields an incorrect answer with no error message.

It is fairly easy to see $\text{numbcomp}(n, m) = \text{binomial}(n-1, m-1)$.

We can obtain random partitions in two different ways (depending on the packages loaded).

```

> draw(Partition(6), size=3);
      [1, 2, 3]
> randpart(6);
      [2, 4]

```

Note in `randpart()` we can not specify the size. It works like

```

> draw(Partition(6), size=allsizes);
          [1, 1, 1, 1, 1, 1]

```

Here is a sample application of compositions.

Suppose we want to extremize $5x^2+4y^3+5z^2$ over the solutions of $x + y + z=82$, with x, y and z all nonzero. Because of the weights (coefficients) we will be dealing with compositions of length 3. How many?

```

> N:=82;
> count(Composition(N), size=3);
          3240

```

We can list them explicitly or we can use an iterator.

```

> w:=lst->5*lst[1]^2+4*lst[2]^3+5*lst[3]^2;
           $w := lst \rightarrow 5 lst_1^2 + 4 lst_2^3 + 5 lst_3^2$ 

```

Initialize minpt, maxpt, minval, maxval fairly arbitrarily -

```

> minpt:=draw(Composition(N), size=3);
          minpt := [28, 36, 18]
> minval:=w(minpt);
          minval := 192164
> maxpt:=draw(Composition(N), size=3);
          maxpt := [28, 21, 33]
> maxval:=w(maxpt);
          maxval := 46409

```

Set up an iterator

```

> iter3:=iterstructs(Composition(N), size=3):
> while not finished(iter3) do
>   newpt:=nextstruct(iter3);
>   newval:=w(newpt);
>   if newval < minval then
>     minpt:=newpt;
>     minval:=w(newpt);
>   fi;
>   if newval > maxval then
>     maxpt:=newpt;

```

```

> maxval:=w(newpt);
> fi;
> od:
>
> minpt, minval;
                                     [38, 6, 38], 15304
> maxpt, maxval;
                                     [1, 80, 1], 2048010

```

So we have a maximum at [1,80,1] and a minimum at [38,6,38].

For larger N this simple method rapidly becomes impractical!

Subsets

```

> allstructs(Subset({a,b,c,c}), size=2);
                                     {{a,b},{a,c},{b,c}}

```

Actually Subset is just another name for Combination.

```

> allstructs(Subset({a,b,c,c}), size=allsizes);
                                     {{ }, {a,b}, {a,c}, {b,c}, {a,b,c}, {c}, {a}, {b}}
> count(Subset({a,b,c,c}), size=allsizes);
                                     8

```

Powerset

The function call powerset(s) returns the set of all the subsets of s if s is a set. If s is an integer then the set of all subsets of {1,...,s} is returned. If s is a list then the list of all sublists of s is returned.

```

> powerset({a,b,c});
                                     {{ }, {a,b}, {a,c}, {b,c}, {a,b,c}, {c}, {a}, {b}}
> powerset([a,b,c]);
                                     [[ ], [a], [b], [a,b], [c], [a,c], [b,c], [a,b,c]]
> powerset({a,b,c,c});
                                     {{ }, {a,b}, {a,c}, {b,c}, {a,b,c}, {c}, {a}, {b}}
> powerset([a,b,c,c]);
                                     [[ ], [c], [c,c], [b], [b,c], [b,c,c], [a], [a,c], [a,c,c], [a,b], [a,b,c], [a,b,c,c]]

```

Note sublists respect order and multiplicities.

We can iterate over the power set by using the iterator returned by the function `subsets()`.

```
> iterpow:=subsets({a,b,c}):  
> while not iterpow[finished] do iterpow[nextvalue] () od;  
      { }  
      { a }  
      { b }  
      { c }  
      { a, b }  
      { a, c }  
      { b, c }  
      { a, b, c }
```

Of course you will want to do something exciting inside the do loop.

Stirling Numbers of the second kind

The function call `stirling2(n,m)` returns $S(n,m)$, the Stirling number of the second kind. $S(n,m)$ is the number of ways of partitioning a set of cardinality n into m (non-empty of course) subsets.

```
> stirling2(6,4);  
      65
```

Bell Numbers and Partitions of Sets

By convention there is one partition of the empty set (the empty partition) with no subsets, but for a non-empty set A partitions consist of non-empty sets (only) with union A .

The Bell number `bell(n)` is the total number of partitions of a set of cardinality n into any number of subsets.

```
> bell(4);  
      15  
> sum(stirling2(4,k),k=1..4);  
      15
```

Be careful to distinguish the notion of partitions of a set and partitions of an integer.

We can easily write a routine to list the partitions of the set $\{1,2,3,\dots,n\}$

```

> parts:=proc(n)
>   local P,A,B,C; option remember;
>   if n=1 then
>     P:={{1}}
>   elif n=2 then
>     P:={ {{1},{2}}, {{1,2}} };
>   elif n>2 then
>     P:={};
>     for A in parts(n-1) do
>       P:= P union { A union {{n}} };
>       for B in A do
>         C:= ( A minus {B} ) union { B union {n} };
>         P:= P union { C };
>       od;
>     od;
>   else
>     P:= FAIL;
>   fi;
>   return P;
> end:

```

Here is an example

```

> parts(4);
{{{2}, {3}, {1, 4}}, {{4}, {1}, {2, 3}}, {{1, 4}, {2, 3}}, {{2, 3, 4}, {1}},
  {{2}, {4}, {1, 3}}, {{1, 3}, {2, 4}}, {{2}, {1, 3, 4}}, {{3}, {4}, {1, 2}},
  {{1, 2}, {3, 4}}, {{3}, {1, 2, 4}}, {{1, 2, 3}, {4}}, {{1, 2, 3, 4}}, {{2}, {3}, {4}, {1}},
  {{3}, {1}, {2, 4}}, {{2}, {1}, {3, 4}}}

```

If you prefer an easier to read list you can try

```

> for A in parts(4) do print(A); od;
      {{2}, {3}, {1, 4}}
      {{4}, {1}, {2, 3}}
      {{1, 4}, {2, 3}}
      {{2, 3, 4}, {1}}
      {{2}, {4}, {1, 3}}
      {{1, 3}, {2, 4}}
      {{2}, {1, 3, 4}}
      {{3}, {4}, {1, 2}}

```

```

{{1, 2}, {3, 4}}
{{3}, {1, 2, 4}}
{{1, 2, 3}, {4}}
{{1, 2, 3, 4}}
{{2}, {3}, {4}, {1}}
{{3}, {1}, {2, 4}}
{{2}, {1}, {3, 4}}

```

More Refined Partitions

Let A be a set of cardinality n .

The number of partitions of A into (nonempty) subsets is $\text{bell}(n)$.

The number of partitions of A into m (nonempty) subsets is $\text{stirling2}(n, m)$.

Now consider the number of partitions of A into m subsets A_i , $i = 1, \dots, m$, where A_i has cardinality n_i . Let's order the set of sets A_i by cardinality to obtain a list. The number of such lists is $\text{multinomial}(n, n_1, \dots, n_m)$. If the cardinalities n_1, \dots, n_m are all distinct then each list corresponds to one partition.

If, on the other hand, N_1, \dots, N_k are the distinct values for the cardinalities n_1, \dots, n_m and each N_i occurs p_i times (we call the p_i the multiplicities) then re-arranging the parts of the list corresponding to a given N_i does not change the corresponding partition and so we have to divide by $(p_i)!$ for each i to get the right count. Thus

If N_1, \dots, N_k are distinct, the number of partitions of a set A of cardinality $n = p_1 * N_1 + \dots + p_k * N_k$ into partitions consisting of p_i sets of cardinality N_i , $i = 1, \dots, k$, is

$$\text{multinomial}(n_1, \dots, n_m) / ((p_1)! \dots (p_k)!)$$

where n_1, \dots, n_m is the sequence of N_i 's with each repeated according to multiplicity. Here's a procedure to compute this number

```

> numbparts:=proc()
>   local k,h,C,M,n,p,allcard;
>   if nargs < 2 then
>     return FAIL;
>   else
>     C:=[];
>     M:=[];

```

```

> n:=0; p:=1;
> for k from 1 to nargs do
>   C:=[op(C),args[k][1]];
>   M:=[op(M),args[k][2]];
>   n:= n + args[k][1] * args[k][2];
>   p:= p * (args[k][2])!
> od;
> allcard:=[];
> for k from 1 to nargs do
>   for h from 1 to M[k] do
>     allcard:=[op(allcard),C[k]];
>   od;
> od;
> return multinomial(n,op(allcard))/p;
> fi;
> end:

```

The input consists of pairs $[N_1, p_1], [N_2, p_2], \dots$ where the N_k are the subset cardinalities and the p_k are the multiplicities.

Here's an example

```

> numbparts( [1,2], [3,1] );

```

10

is the number of partitions of the set $\{1,2,3,4,5\}$ consisting of 3 subsets, 2 of which have 1 element and the third of which, has 3 elements. Let's check it.

```

> for A in parts(5) do if nops(A)=3 then
> if nops(A[1])=3 or nops(A[2])=3 or nops(A[3])=3
> then print(A); fi; fi; od;

```

$\{\{2\}, \{3, 4, 5\}, \{1\}\}$
 $\{\{3\}, \{2, 4, 5\}, \{1\}\}$
 $\{\{3\}, \{1, 2, 5\}, \{4\}\}$
 $\{\{3\}, \{1, 2, 4\}, \{5\}\}$
 $\{\{1, 2, 3\}, \{4\}, \{5\}\}$
 $\{\{2\}, \{1, 3, 4\}, \{5\}\}$
 $\{\{2, 3, 4\}, \{1\}, \{5\}\}$
 $\{\{2\}, \{3\}, \{1, 4, 5\}\}$
 $\{\{4\}, \{2, 3, 5\}, \{1\}\}$
 $\{\{2\}, \{1, 3, 5\}, \{4\}\}$

Sure enough there are 10 of them.

Generating Functions

```
> x:='x';
```

```
x := x
```

In dealing with generating functions we often need to extract a certain coefficient of a polynomial. Here is a function which performs this operation

```
> getc := (p, x, n) -> coeff(convert(taylor(p, x=0, n+1), 'polynom'), x, n);  
getc := (p, x, n) → coeff(convert(taylor(p, x = 0, n + 1), polynom), x, n)
```

Suppose we have 7 blue marbles, 8 green marbles and 8 red marbles in an urn (they are always in an urn). How many ways can we draw out 14 marbles consisting of an odd number of blue marbles, an even number of green marbles and a prime number of red marbles (and with no regard for order) and at least one marble of each color?.

The generating function is

```
> p := (x+x^3+x^5+x^7) * (x^2+x^4+x^6+x^8) * (x^2+x^3+x^5+x^7);  
p := (x + x3 + x5 + x7) (x2 + x4 + x6 + x8) (x2 + x3 + x5 + x7)  
> getc(p, x, 14);
```

```
10
```

This is a small enough number that you can try to enumerate the possibilities.

We can use a simple modification of `getc()` for exponential generating functions:

```
> getxc := (p, x, n) -> (n!) * getc(p, x, n);  
getxc := (p, x, n) → n! getc(p, x, n)
```

We have 3 red, 3 blue and 3 white marbles (in an urn of course). How many ways can we arrange 5 of them in a row so that each color is used at least once (order matters)?

```
> q := (x + (1/2) * x^2 + (1/6) * x^3) ^ 3;
```

$$q := \left(x + \frac{1}{2}x^2 + \frac{1}{6}x^3 \right)^3$$

```
> getxc(q, x, 5);
```

```
150
```

The number of m -compositions of an integer n has generating function (expression)

```
> g := (Sum(x^k, k=1..infinity))^m = (x/(1-x))^m;
```

$$g := \left(\sum_{k=1}^{\infty} x^k \right)^m = \left(\frac{x}{1-x} \right)^m$$

So the number of compositions of length 4 of 12 is given by

```
> getc((x/(1-x))^4, x, 12);
```

165

Let's check it:

```
> count(Composition(12), size=4);
```

165

Here's another

```
> getc((x/(1-x))^15, x, 22);
```

116280

```
> count(Composition(22), size=15);
```

116280

Sure enough!

Graycode

The function call `graycode(n)` returns the list of all integers from 0 to 2^n-1 in graycode order, that is, so each successive pair of integers when expressed in binary differ by only one bit.

```
> gco:=graycode(4);
```

```
gco := [0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8]
```

```
> for k in gco do printf("\n%04d", convert(k,binary)); od;
```

```
0000
0001
0011
0010
0110
0111
0101
0100
1100
```

1101
1111
1110
1010
1011
1001
1000

Traditional Problems

Some of these problems make use of the concepts above and some do not.

1. How many ways can a committee of 5 women and 4 men be selected from an organization of 12 women and 14 men?
2. How many strings of length 10 can be formed from 4 a's, 3 b's and 3 c's?
3. If a fair coin is tossed 10 times how many ways are there of getting exactly 5 heads?
4. A set has cardinality 12. How many subsets have cardinality 4?
5. A set has cardinality 12. How many partitions have 4 (nonempty) sets?
6. How many 7 (decimal) digit positive numbers have all their digits different?
7. How many 7 (decimal) digit positive numbers have at least 2 digits different?
8. How many 7 (decimal) digit positive numbers contain 3 distinct pairs of equal digits?
9. How many integers between 397 and 10483 are divisible either by 7 or 11?
10. How many 6 digit integers have 3, 4 or 9 as a digit?
11. How many distinct partitions of length 3, 4 or 5 are there of 32?
12. How many ways can you permute the letters of MISSISSIPPI?
13. An urn contains 6 red marble, 7 blue marbles and 11 green marbles. How many ways can you select 9 marbles so each color occurs at least once, but no more than 5 times? (Hint: Try using a generating function.)
14. A partition of a positive integer n is a list of positive integers with sum n . Two partitions are considered identical if they differ only in order. A partition is said to be distinct if the integers in it are distinct. Find the number of distinct partitions of 10.

[15. The number of partitions of n is the coefficient of x^n in the generating function

[> `k:='k' : x:='x' :g:=product ((1-x^k)^(-1),k=1..n) ;`

$$g := \prod_{k=1}^n \frac{1}{1-x^k}$$

[16. The number of distinct partitions of n is the coefficient of x^n in the generating function

[> `gd:=product ((1+x^k),k=1..n) ;`

$$gd := \prod_{k=1}^n (1+x^k)$$

[In problems 15 and 16 we really should use (formal) infinite products for the generating functions.

[>