

# Cartesian Product of a Finite Number of Sets

Mth 355/399 Oct 10 2001 Maple 6

Bent E. Petersen

Filename: 355f2001-cartesian-product.mws

Supplement to

[Set Theory - Maple Programming Assignment](#)

Mth 355 Fall 2001 Assignment 2. Due Oct 12.

Mth 355/399 Oct 5 2001 Maple 6

Bent E. Petersen

Filename: 355f2001-assign-002-maple-prog.mws

In assignment 2 I gave you a procedure to compute the Cartesian product of two sets and asked you to write a procedure to compute the Cartesian product of any number of sets (problem 3). I did give you a hint how you could handle 3 sets. This problem created quite a bit of difficulty!

In this note I will present 2 solutions - a recursive solution and a nonrecursive one. I left plenty of room for improvement in case you wanted to try your hand at improving my code.

Here is the original carp() procedure for 2 sets.

```
> carp:=proc (X,Y)
>   local Z,x,y;
>   Z:={};
>   for x in X do
>     for y in Y do
>       Z:=Z union {[x,y]};
>     od;
>   od;
>   return Z;
> end;
```

Let's test it on some test sets

```
> A1:={1,3,5,9}; A2:={12,13,15}; A3:={2,4,6}; A4:={5,9};
      A1 := {1, 3, 5, 9}
      A2 := {12, 13, 15}
      A3 := {2, 4, 6}
      A4 := {5, 9}
> carp(A1,A2);
[[1, 12], [1, 13], [1, 15], [3, 12], [3, 13], [3, 15], [5, 12], [5, 13], [5, 15], [9, 12], [9, 13],
```

[ 9, 15 ] }

According to the hint I gave you we can use `carp()` to compute a triple product

```
> Z:={}: for x in carp(A1,A2) do for a in A3 do Z:=Z union
  { [op(x),a] }; od; od; Z;
[ 5, 13, 2 ], [ 5, 13, 4 ], [ 5, 15, 2 ], [ 5, 12, 4 ], [ 5, 12, 6 ], [ 5, 13, 6 ], [ 1, 12, 2 ], [ 1, 12, 4 ],
  [ 1, 12, 6 ], [ 1, 13, 2 ], [ 1, 13, 4 ], [ 1, 13, 6 ], [ 1, 15, 2 ], [ 1, 15, 4 ], [ 1, 15, 6 ], [ 3, 12, 2 ],
  [ 3, 12, 4 ], [ 3, 12, 6 ], [ 3, 13, 2 ], [ 3, 13, 4 ], [ 3, 13, 6 ], [ 3, 15, 2 ], [ 3, 15, 4 ], [ 3, 15, 6 ],
  [ 5, 12, 2 ], [ 5, 15, 4 ], [ 5, 15, 6 ], [ 9, 12, 2 ], [ 9, 12, 4 ], [ 9, 12, 6 ], [ 9, 13, 2 ], [ 9, 13, 4 ],
  [ 9, 13, 6 ], [ 9, 15, 2 ], [ 9, 15, 4 ], [ 9, 15, 6 ] }
```

Once we have a triple product we can use the same process to do a quadruple product. It only remains to do this successive multiplication either by recursion or by a loop over the arguments (the factors).

Now lets write a procedure `mcarp()` to compute the Cartesian product of any number of sets.

## Recursive `mcarp()`

This recursive follows the hint I gave you.

```
> mcarp:=proc()
>   local Z,k,x,y; option remember;
>   if nargs=0 then
>     Z:={};
>   elif nargs=1 then
>     Z:=args[1];
>   else Z:={};
>     for x in mcarp( seq(args[k], k=1..nargs-1) ) do
>       for y in args[nargs] do
>         Z:= Z union { [op(x),y] };
>       od;
>     od;
>   fi;
>   return Z;
> end;
```

The "option remember" is not really needed but it speeds up recursive routines by a great deal (in exchange for using much more RAM).

Note how the routine recurses - to compute the product of  $n$  sets it calls itself to find the product of

n-1 sets and then uses the hint to take the product with an additional set.

Let's test our `mcarp()`:

```
> mcarp(A1,A2);
{[1, 12], [1, 13], [1, 15], [3, 12], [3, 13], [3, 15], [5, 12], [5, 13], [5, 15], [9, 12], [9, 13],
 [9, 15]}
> mcarp(A1,A2,A3);
{[5, 13, 2], [5, 13, 4], [5, 15, 2], [5, 12, 4], [5, 12, 6], [5, 13, 6], [1, 12, 2], [1, 12, 4],
 [1, 12, 6], [1, 13, 2], [1, 13, 4], [1, 13, 6], [1, 15, 2], [1, 15, 4], [1, 15, 6], [3, 12, 2],
 [3, 12, 4], [3, 12, 6], [3, 13, 2], [3, 13, 4], [3, 13, 6], [3, 15, 2], [3, 15, 4], [3, 15, 6],
 [5, 12, 2], [5, 15, 4], [5, 15, 6], [9, 12, 2], [9, 12, 4], [9, 12, 6], [9, 13, 2], [9, 13, 4],
 [9, 13, 6], [9, 15, 2], [9, 15, 4], [9, 15, 6]}
> mcarp(A1,A2,A3,A4);
{[3, 15, 4, 9], [9, 15, 6, 9], [1, 13, 4, 9], [9, 15, 6, 5], [3, 15, 6, 9], [1, 13, 2, 9], [3, 15, 6, 5],
 [1, 13, 4, 5], [3, 15, 2, 9], [3, 15, 4, 5], [1, 13, 2, 5], [5, 13, 2, 5], [5, 13, 2, 9], [5, 13, 4, 5],
 [5, 13, 4, 9], [5, 15, 2, 5], [5, 15, 2, 9], [5, 12, 4, 5], [5, 12, 4, 9], [5, 12, 6, 5], [5, 12, 6, 9],
 [5, 13, 6, 5], [5, 13, 6, 9], [1, 12, 2, 5], [1, 12, 2, 9], [1, 12, 4, 5], [1, 12, 4, 9], [1, 12, 6, 5],
 [1, 12, 6, 9], [1, 13, 6, 5], [1, 13, 6, 9], [1, 15, 2, 5], [1, 15, 2, 9], [1, 15, 4, 5], [1, 15, 4, 9],
 [1, 15, 6, 5], [1, 15, 6, 9], [3, 12, 2, 5], [3, 12, 2, 9], [3, 12, 4, 5], [3, 12, 4, 9], [3, 12, 6, 5],
 [3, 12, 6, 9], [3, 13, 2, 5], [3, 13, 2, 9], [3, 13, 4, 5], [3, 13, 4, 9], [3, 13, 6, 5], [3, 13, 6, 9],
 [3, 15, 2, 5], [5, 12, 2, 5], [5, 12, 2, 9], [5, 15, 4, 5], [5, 15, 4, 9], [5, 15, 6, 5], [5, 15, 6, 9],
 [9, 12, 2, 5], [9, 12, 2, 9], [9, 12, 4, 5], [9, 12, 4, 9], [9, 12, 6, 5], [9, 12, 6, 9], [9, 13, 2, 5],
 [9, 13, 2, 9], [9, 13, 4, 5], [9, 13, 4, 9], [9, 13, 6, 5], [9, 13, 6, 9], [9, 15, 2, 5], [9, 15, 2, 9],
 [9, 15, 4, 5], [9, 15, 4, 9]}
```

## Nonrecursive `mcarp()`

A nonrecursive `mcarp()` is harder to write, but easier to understand and will probably run more efficiently. Let's call this version `mcarp2()`.

```
> mcarp2:=proc()
>   local Z,U,k,x,y;
>   if nargs=0 then
>     Z:={};
>   elif nargs=1 then
>     Z:=args[1];
>   else Z:=args[1];
>   for k from 2 to nargs do
```

```

> U:={};
> for x in Z do
>   for y in args[k] do
>     U:= U union {[op(x),y]};
>   od;
> od;
> Z:=U;
> od;
> fi;
> return Z;
> end:

```

Let's test it -

```

> mcarp2 (A1,A2) ;

```

```

{[1, 12], [1, 13], [1, 15], [3, 12], [3, 13], [3, 15], [5, 12], [5, 13], [5, 15], [9, 12], [9, 13],
 [9, 15]}

```

```

> mcarp2 (A1,A2,A3) ;

```

```

{[5, 13, 2], [5, 13, 4], [5, 15, 2], [5, 12, 4], [5, 12, 6], [5, 13, 6], [1, 12, 2], [1, 12, 4],
 [1, 12, 6], [1, 13, 2], [1, 13, 4], [1, 13, 6], [1, 15, 2], [1, 15, 4], [1, 15, 6], [3, 12, 2],
 [3, 12, 4], [3, 12, 6], [3, 13, 2], [3, 13, 4], [3, 13, 6], [3, 15, 2], [3, 15, 4], [3, 15, 6],
 [5, 12, 2], [5, 15, 4], [5, 15, 6], [9, 12, 2], [9, 12, 4], [9, 12, 6], [9, 13, 2], [9, 13, 4],
 [9, 13, 6], [9, 15, 2], [9, 15, 4], [9, 15, 6]}

```

```

> mcarp2 (A1,A2,A3,A4) ;

```

```

{[3, 15, 4, 9], [9, 15, 6, 9], [1, 13, 4, 9], [9, 15, 6, 5], [3, 15, 6, 9], [1, 13, 2, 9], [3, 15, 6, 5],
 [1, 13, 4, 5], [3, 15, 2, 9], [3, 15, 4, 5], [1, 13, 2, 5], [5, 13, 2, 5], [5, 13, 2, 9], [5, 13, 4, 5],
 [5, 13, 4, 9], [5, 15, 2, 5], [5, 15, 2, 9], [5, 12, 4, 5], [5, 12, 4, 9], [5, 12, 6, 5], [5, 12, 6, 9],
 [5, 13, 6, 5], [5, 13, 6, 9], [1, 12, 2, 5], [1, 12, 2, 9], [1, 12, 4, 5], [1, 12, 4, 9], [1, 12, 6, 5],
 [1, 12, 6, 9], [1, 13, 6, 5], [1, 13, 6, 9], [1, 15, 2, 5], [1, 15, 2, 9], [1, 15, 4, 5], [1, 15, 4, 9],
 [1, 15, 6, 5], [1, 15, 6, 9], [3, 12, 2, 5], [3, 12, 2, 9], [3, 12, 4, 5], [3, 12, 4, 9], [3, 12, 6, 5],
 [3, 12, 6, 9], [3, 13, 2, 5], [3, 13, 2, 9], [3, 13, 4, 5], [3, 13, 4, 9], [3, 13, 6, 5], [3, 13, 6, 9],
 [3, 15, 2, 5], [5, 12, 2, 5], [5, 12, 2, 9], [5, 15, 4, 5], [5, 15, 4, 9], [5, 15, 6, 5], [5, 15, 6, 9],
 [9, 12, 2, 5], [9, 12, 2, 9], [9, 12, 4, 5], [9, 12, 4, 9], [9, 12, 6, 5], [9, 12, 6, 9], [9, 13, 2, 5],
 [9, 13, 2, 9], [9, 13, 4, 5], [9, 13, 4, 9], [9, 13, 6, 5], [9, 13, 6, 9], [9, 15, 2, 5], [9, 15, 2, 9],
 [9, 15, 4, 5], [9, 15, 4, 9]}

```

```

>

```