# Discrete Mathematics Project

Joshua Beninson

## Cartesian Product Program

In order to derive an appropriate method of outputting the Cartesian Product of a large number of sets that consisted of a large number of elements, the definition of the Cartesian Product was consulted.

Let <Sn> be a sequence of sets.
The **cartesian product** of $\langle S_n \rangle$ is defined as:

$$\prod_{k=1}^{n} S_k = \{(x_1, x_2, \ldots, x_n) : \forall k \in \mathbb{N}_n^* : x_k \in S_k\}$$

The Cartesian Product is also denoted by $S_1 \times S_2 \times \ldots \times S_n$.
Thus, $S_1 \times S_2 \times \ldots \times S_n$ is the set of all ordered n-tuples $(x_1, x_2, \ldots, x_n)$ with $x_k \in S_k$.

Each individual set's input stream is parsed and inserted into an array. The commas were used as delimiters to tokenize the stream. Those tokens were then inserted into each individual set's array as its elements. A two dimensional array was used where the rows represent each set, and the columns represent each element of the set.

### Input Streams Example:

Set 1: aplewf, fefw, rfewfw, sdfdsf, gsd, …., dfvsdf
Set 2: dfvsdf, jhefwe, lpiefwe, efwef, dsfsf,…., dfsdf
Etc…

### How It Is Stored In The System:

| | element 1 | element 2 | element 3 | element 4 | element 5 | …. | element n |
|---|---|---|---|---|---|---|---|
| **set 1** | aplewf | fefw | rfewfw | sdfdsf | gsd | …. | dfvsdf |
| **set 2** | dfvsdf | jhefwe | lpiefwe | efwef | dsfsf | …. | dfsdf |
| **…** | …. | …. | …. | …. | …. | …. | …. |
| **set n** | adfewf | fadgarg | asdefqe | wwefrt | dfsdfsdf | …. | asdfsa |

As each set was being entered into the array, an integer array (called: elements_per_set) was used to hold the total number of elements that were entered into a set. The first element in this array would correspond to the total number of elements entered into the first set, the second element would correspond to the total number of elements entered into the second set, and so on. In other words, if 5 elements were entered into Set 1, elements_per_set[Set 1] would equal 5. (The actual code would be elements_per_set [0] = 5, since we will use the convention of arrays beginning at an index 0.)

The algorithm additionally calculates the number of times the elements of each set was printed and in what order. Representing the sets and the corresponding elements in a two dimensional array, it became apparent that there was a connection between the number of elements in each set, the number of sets,

and the order of the set's n-tuples.

The amount of times that each element of each respective array was printed and in what order is directly related to its position in the equation and the amount of elements in each set that follow it. For example, if you have 5 sets of 5 elements, every element of the first set will be printed 1*5*5*5*5 times before outputting the next element in the set. An element of the 3rd set would be printed 1*5*5 times before changing the element to output, and so forth. Thus, to find the total number of times an element in a particular set will be present in an ordered pair, we multiply the number of elements of that were input after each particular set. As defined here, this number is stored in the integer array output_ticker[i], where index i represents the set the variable is referring to. In the event that any of the sets are found to be a NULL set, the product is immediately output as { } , and the program exits.

```
/*Calulates the amount of prints that must be completed before iterating to the next
	 * element in the set for each set and enters it into each sets spot in the output_ticker
	 * array*/
	int j=0;
	for(i=0;i<sets;i++){
		output_ticker[i]=1;
		for(j=(i+1);j<sets;j++){
			output_ticker[i]*=elements_per_set[j];
		}
		if(elements_per_set[i]==0){
			printf("{ }\n\n");
			return 0;
		}
	}
```

The traversal equation uses the number of ordered pairs, that have been outputted so far (this value is stored in the integer value:print), divided by the output_ticker as previously explained. The mod of this result and the elements_per_set of the respective set are then used to calculate the exact element in each set to print in each output set iteration.

```
/*This loop traverses each set during each print iteration and finds the appropriate element to output in
*each ordered pair based on the integer output of (print divided by the output ticker of the set that is
being considered) and then finding the mod of that result with respect to the elements in that set in
order to not traverse past the entered values in each set*/

int print=0;
	printf("{ ");
	for(print=0;print<(output_ticker[0]*elements_per_set[0]);print++){
		printf("( ");
			for(i=0;i<sets;i++){
				element=(print/output_ticker[i])%elements_per_set[i];

				printf("%s",cart[i][element]);
```

```
                        if(i<(sets-1))
                                printf(", ");
                }
        printf(" )");
        if(print<(output_ticker[0]*elements_per_set[0])-1)
                printf(",\n");
}
printf(" }\n\n");
```

SAMPLE OUTPUT FROM LINUX TERMINAL

```
josh@ubuntu:~/Dropbox$ gcc -g Cartesian.c -o Cartesian

josh@ubuntu:~/Dropbox$ ./Cartesian


Please input the number of sets you wish to find the Cartesian Product for:

6


Each set can handle up to 1000 elements.

Please input elements of up to nine characters into each set.

Each element should be separated by commas.

Press <ENTER> to complete input for a designated set.

Press <Enter> as the first element of a set for a NULL set.....


Set 1: 1,65756,234234,58768,342

Set 2: 234,34,456779,346124

Set 3: 213,324

Set 4: 789789589,5673452345,23423

Set 5:

Set 6: 3421,57,2341234
```

Set 1 x Set 2 x Set 3 x Set 4 x Set 5 x Set 6 =

{ }

josh@ubuntu:~/Dropbox$

josh@ubuntu:~/Dropbox$ ./Cartesian

Please input the number of sets you wish to find the Cartesian Product for:

2

Each set can handle up to 1000 elements.

Please input elements of up to nine characters into each set.

Each element should be separated by commas.

Press <ENTER> to complete input for a designated set.

Press <Enter> as the first element of a set for a NULL set.....

Set 1: 567857,12342,234534565

Set 2: 23451,234123547

Set 1 x Set 2 =

{ ( 567857, 23451 ),

( 567857, 234123547 ),

( 12342, 23451 ),

( 12342, 234123547 ),

( 234534565, 23451 ),

( 234534565, 234123547 ) }

josh@ubuntu:~/Dropbox$

josh@ubuntu:~/Dropbox$ ./Cartesian

Please input the number of sets you wish to find the Cartesian Product for:

4

Each set can handle up to 1000 elements.

Please input elements of up to nine characters into each set.

Each element should be separated by commas.

Press <ENTER> to complete input for a designated set.

Press <Enter> as the first element of a set for a NULL set.....

Set 1: 1,2,3,4

Set 2: 5,6,7,8

Set 3: 9,10,11,12

Set 4: 13,14,15,16

Set 1 x Set 2 x Set 3 x Set 4 =

{ ( 1, 5, 9, 13 ),

( 1, 5, 9, 14 ),

( 1, 5, 9, 15 ),

( 1, 5, 9, 16 ),

( 1, 5, 10, 13 ),

( 1, 5, 10, 14 ),

( 1, 5, 10, 15 ),

( 1, 5, 10, 16 ),

( 1, 5, 11, 13 ),

( 1, 5, 11, 14 ),

( 1, 5, 11, 15 ),

( 1, 5, 11, 16 ),

( 1, 5, 12, 13 ),

( 1, 5, 12, 14 ),

( 1, 5, 12, 15 ),

( 1, 5, 12, 16 ),

( 1, 6, 9, 13 ),

( 1, 6, 9, 14 ),

( 1, 6, 9, 15 ),

( 1, 6, 9, 16 ),

( 1, 6, 10, 13 ),

( 1, 6, 10, 14 ),

( 1, 6, 10, 15 ),

( 1, 6, 10, 16 ),

( 1, 6, 11, 13 ),

( 1, 6, 11, 14 ),

( 1, 6, 11, 15 ),

( 1, 6, 11, 16 ),

( 1, 6, 12, 13 ),

( 1, 6, 12, 14 ),

( 1, 6, 12, 15 ),

( 1, 6, 12, 16 ),

( 1, 7, 9, 13 ),

( 1, 7, 9, 14 ),

( 1, 7, 9, 15 ),

( 1, 7, 9, 16 ),

( 1, 7, 10, 13 ),

( 1, 7, 10, 14 ),

( 1, 7, 10, 15 ),

( 1, 7, 10, 16 ),

( 1, 7, 11, 13 ),

( 1, 7, 11, 14 ),

( 1, 7, 11, 15 ),

( 1, 7, 11, 16 ),

( 1, 7, 12, 13 ),

( 1, 7, 12, 14 ),

( 1, 7, 12, 15 ),

( 1, 7, 12, 16 ),

( 1, 8, 9, 13 ),

( 1, 8, 9, 14 ),

( 1, 8, 9, 15 ),

( 1, 8, 9, 16 ),

( 1, 8, 10, 13 ),

( 1, 8, 10, 14 ),

( 1, 8, 10, 15 ),

( 1, 8, 10, 16 ),

( 1, 8, 11, 13 ),

( 1, 8, 11, 14 ),

( 1, 8, 11, 15 ),

( 1, 8, 11, 16 ),

( 1, 8, 12, 13 ),

( 1, 8, 12, 14 ),

( 1, 8, 12, 15 ),

( 1, 8, 12, 16 ),

( 2, 5, 9, 13 ),

( 2, 5, 9, 14 ),

( 2, 5, 9, 15 ),

( 2, 5, 9, 16 ),

( 2, 5, 10, 13 ),

( 2, 5, 10, 14 ),

( 2, 5, 10, 15 ),

( 2, 5, 10, 16 ),

( 2, 5, 11, 13 ),

( 2, 5, 11, 14 ),

( 2, 5, 11, 15 ),

( 2, 5, 11, 16 ),

( 2, 5, 12, 13 ),

( 2, 5, 12, 14 ),

( 2, 5, 12, 15 ),

( 2, 5, 12, 16 ),

( 2, 6, 9, 13 ),

( 2, 6, 9, 14 ),

( 2, 6, 9, 15 ),

( 2, 6, 9, 16 ),

( 2, 6, 10, 13 ),

( 2, 6, 10, 14 ),

( 2, 6, 10, 15 ),

( 2, 6, 10, 16 ),

( 2, 6, 11, 13 ),

( 2, 6, 11, 14 ),

( 2, 6, 11, 15 ),

( 2, 6, 11, 16 ),

( 2, 6, 12, 13 ),

( 2, 6, 12, 14 ),

( 2, 6, 12, 15 ),

( 2, 6, 12, 16 ),

( 2, 7, 9, 13 ),

( 2, 7, 9, 14 ),

( 2, 7, 9, 15 ),

( 2, 7, 9, 16 ),

( 2, 7, 10, 13 ),

( 2, 7, 10, 14 ),

( 2, 7, 10, 15 ),

( 2, 7, 10, 16 ),

( 2, 7, 11, 13 ),

( 2, 7, 11, 14 ),

( 2, 7, 11, 15 ),

( 2, 7, 11, 16 ),

( 2, 7, 12, 13 ),

( 2, 7, 12, 14 ),

( 2, 7, 12, 15 ),

( 2, 7, 12, 16 ),

( 2, 8, 9, 13 ),

( 2, 8, 9, 14 ),

( 2, 8, 9, 15 ),

( 2, 8, 9, 16 ),

( 2, 8, 10, 13 ),

( 2, 8, 10, 14 ),

( 2, 8, 10, 15 ),

( 2, 8, 10, 16 ),

( 2, 8, 11, 13 ),

( 2, 8, 11, 14 ),

( 2, 8, 11, 15 ),

( 2, 8, 11, 16 ),

( 2, 8, 12, 13 ),

( 2, 8, 12, 14 ),

( 2, 8, 12, 15 ),

( 2, 8, 12, 16 ),

( 3, 5, 9, 13 ),

( 3, 5, 9, 14 ),

( 3, 5, 9, 15 ),

( 3, 5, 9, 16 ),

( 3, 5, 10, 13 ),

( 3, 5, 10, 14 ),

( 3, 5, 10, 15 ),

( 3, 5, 10, 16 ),

( 3, 5, 11, 13 ),

( 3, 5, 11, 14 ),

( 3, 5, 11, 15 ),

( 3, 5, 11, 16 ),

( 3, 5, 12, 13 ),

( 3, 5, 12, 14 ),

( 3, 5, 12, 15 ),

( 3, 5, 12, 16 ),

( 3, 6, 9, 13 ),

( 3, 6, 9, 14 ),

( 3, 6, 9, 15 ),

( 3, 6, 9, 16 ),

( 3, 6, 10, 13 ),

( 3, 6, 10, 14 ),

( 3, 6, 10, 15 ),

( 3, 6, 10, 16 ),

( 3, 6, 11, 13 ),

( 3, 6, 11, 14 ),

( 3, 6, 11, 15 ),

( 3, 6, 11, 16 ),

( 3, 6, 12, 13 ),

( 3, 6, 12, 14 ),

( 3, 6, 12, 15 ),

( 3, 6, 12, 16 ),

( 3, 7, 9, 13 ),

( 3, 7, 9, 14 ),

( 3, 7, 9, 15 ),

( 3, 7, 9, 16 ),

( 3, 7, 10, 13 ),

( 3, 7, 10, 14 ),

( 3, 7, 10, 15 ),

( 3, 7, 10, 16 ),

( 3, 7, 11, 13 ),

( 3, 7, 11, 14 ),

( 3, 7, 11, 15 ),

( 3, 7, 11, 16 ),

( 3, 7, 12, 13 ),

( 3, 7, 12, 14 ),

( 3, 7, 12, 15 ),

( 3, 7, 12, 16 ),

( 3, 8, 9, 13 ),

( 3, 8, 9, 14 ),

( 3, 8, 9, 15 ),

( 3, 8, 9, 16 ),

( 3, 8, 10, 13 ),

( 3, 8, 10, 14 ),

( 3, 8, 10, 15 ),

( 3, 8, 10, 16 ),

( 3, 8, 11, 13 ),

( 3, 8, 11, 14 ),

( 3, 8, 11, 15 ),

( 3, 8, 11, 16 ),

( 3, 8, 12, 13 ),

( 3, 8, 12, 14 ),

( 3, 8, 12, 15 ),

( 3, 8, 12, 16 ),

( 4, 5, 9, 13 ),

( 4, 5, 9, 14 ),

( 4, 5, 9, 15 ),

( 4, 5, 9, 16 ),

( 4, 5, 10, 13 ),

( 4, 5, 10, 14 ),

( 4, 5, 10, 15 ),

( 4, 5, 10, 16 ),

( 4, 5, 11, 13 ),

( 4, 5, 11, 14 ),

( 4, 5, 11, 15 ),

( 4, 5, 11, 16 ),

( 4, 5, 12, 13 ),

( 4, 5, 12, 14 ),

( 4, 5, 12, 15 ),

( 4, 5, 12, 16 ),

( 4, 6, 9, 13 ),

( 4, 6, 9, 14 ),

( 4, 6, 9, 15 ),

( 4, 6, 9, 16 ),

( 4, 6, 10, 13 ),

( 4, 6, 10, 14 ),

( 4, 6, 10, 15 ),

( 4, 6, 10, 16 ),

( 4, 6, 11, 13 ),

( 4, 6, 11, 14 ),

( 4, 6, 11, 15 ),

( 4, 6, 11, 16 ),

( 4, 6, 12, 13 ),

( 4, 6, 12, 14 ),

( 4, 6, 12, 15 ),

( 4, 6, 12, 16 ),

( 4, 7, 9, 13 ),

( 4, 7, 9, 14 ),

( 4, 7, 9, 15 ),

( 4, 7, 9, 16 ),

( 4, 7, 10, 13 ),

( 4, 7, 10, 14 ),

( 4, 7, 10, 15 ),

( 4, 7, 10, 16 ),

( 4, 7, 11, 13 ),

( 4, 7, 11, 14 ),

( 4, 7, 11, 15 ),

( 4, 7, 11, 16 ),

( 4, 7, 12, 13 ),

( 4, 7, 12, 14 ),

( 4, 7, 12, 15 ),

( 4, 7, 12, 16 ),

( 4, 8, 9, 13 ),

( 4, 8, 9, 14 ),

( 4, 8, 9, 15 ),

( 4, 8, 9, 16 ),

( 4, 8, 10, 13 ),

( 4, 8, 10, 14 ),

( 4, 8, 10, 15 ),

( 4, 8, 10, 16 ),

( 4, 8, 11, 13 ),

( 4, 8, 11, 14 ),

( 4, 8, 11, 15 ),

( 4, 8, 11, 16 ),

( 4, 8, 12, 13 ),

( 4, 8, 12, 14 ),

( 4, 8, 12, 15 ),

( 4, 8, 12, 16 ) }

josh@ubuntu:~/Dropbox$