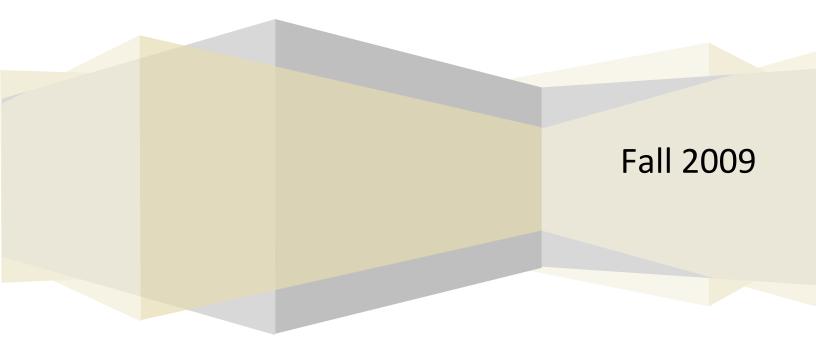
ECSE420 - Parallel Computing

Multi Thread Performance Analysis of a Brute-Force Sudoku Solver

Simon Foucher G07



THE GAME

Sudoku is a popular puzzle game similar to crosswords, but using numbers. The Sudoku grid is a square, divided in subsquares containing as many elements as the length of the row of the main square. Which such a partition, every element in the Sudoku board has 3 constraints: row, column and sub square. The constraint stipulates that a number, ranging from 1 to the size of a subsquare, an only be places once in a row, a column and a sub square.

There are many strategies one can use to solve Sudoku puzzles, and solutions might not be unique, or might not even exist! The difficulty rating of a board is relative to the number of indices provided. A standard Sudoku board contains 81 places, and is subdivided into 9 subsquares each containing 9 elements.

APPROACH

When solved by a computer, one of the most complete approaches to solving a Sudoku is using brute force. This provides 2 advantages over algorithmic solvers: first, it can confirm if there are no solutions, and second, if there are many solutions, the computer can easily output a complete list of them. In this work we have developed a multi-threaded brute force Sudoku solver. To standardize the approach, the same board has been used throughout the development. The puzzle is rated as "Extremely difficult" and has been taken from:

http://en.wikipedia.org/wiki/Algorithmics_of_sudoku

-												-
I.	1	*	*	I.	*	*	*	I.	*	*	2	1
I.	*	9	*	I.	4	*	*	I.	*	5	*	1
I.	*	*	6	I.	*	*	*	I.	7	*	*	1
												-
I.	*	5	×	I.	9	*	3	I.	*	*	*	1
I.	*	*	*	T.	*	7	×	I.	*	*	*	1
I.	*	*	*	I.	8	5	*	I.	*	4	*	1
-												-
I.	7	*	*	I.	*	*	*	I.	6	*	*	1
I.	*	3	*	I.	*	*	9	I.	*	8	*	1
I.	*	*	2	I.	*	*	*	I.	*	*	*	1

Figure 1: the Sudoku puzzle used for gathering the performance data.

The experiment will consist in repeatedly solving this puzzle using the same algorithm with a varying number of threads to observe the variation in performance obtained.

The algorithm uses recursive calls on a test-and-backtrack approach. All the cells of the board will be visited.

 Upon arriving, if the cell is not empty, it is skipped. Otherwise, a temporary value is assigned from 1 to 9. In figure 2, we can observe that the first cell (0,0) got skipped and that the value of '1' has been assigned to the next cell (0,1)

| 1 1 * | * * * | * * 2 | | * 9 * | 4 * * | * 5 * | | * * 6 | * * * | 7 * * | | * 5 * | 9 * 3 | * * * | | * * * | * 7 * | * * * | | * * * | 8 5 * | * 4 * | | 7 * * | * * * | 6 * * | | * 3 * | * * 9 | * 8 * | | * * 2 | * * * | * * * |

Figure2: Testing the legality of a '1' at the position (0,1

- 2. Afterwards, a test is performed to see if the attempted value is allowed there. By scanning the 3 dimensions of neighbors, the new value is compared to all the elements contained in its row, its column and its sub square. If the same element is encountered, the next value is tested and the algorithm repeats the test. If all the values have been exhausted, the algorithm backtracks to the previous cell.
- 3. This process is repeated until the program finds an element that could potentially exist at that location

											-
1	3	×	I.	*	*	*	T.	*	*	2	I.
*	9	*	I.	4	*	*	T.	*	5	*	I.
*	*	6	I.	*	*	*	I.	7	*	*	I
											-
*	5	*	I.	9	*	3	T.	*	*	*	I.
*	*	×	I.	*	7	*	I.	*	*	*	I.
*	*	*	I.	8	5	*	I.	*	4	*	I.
											-
7	*	×	ī.	*	*	*	ī.	6	*	*	ī
*	3	*	I.	*	*	9	T.	*	8	*	I.
*	*	2	I.	*	*	*	T.	*	*	*	I.
											-

Figure 3: '3' is the first 'legal' value we can leave at position (0,1) before moving on to the next position

4. Once this condition is satisfied, the function is recursively called to the next location. To optimize the software, when the function is called a second time, the first element it looks at is

the previous element incremented by 1. For example here, there since we just guessed '3 to be at location (0,1), the algorithm will guess '4' the next location

- 5. Once the depth of the recursion reaches 81, we know that all the cells have been visited, so if no solution was found, we backtrack one cell and keep trying all the possibilities on that one by recalling the recursion.
- 6. This keeps happening until all the possibilities have been tried out. If no solution was bound in the mean time, an error message is printed on the board (for the purpose of this experiment, we know that there is a solution so it is always found sooner or later)

OPTIMIZING BRUTE FORCE

By definition, a brute force algorithm is the least optimal solution to a problem, but guarantees results. The time required to solve the puzzle greatly depends on where the search is conducted, and which elements are visited first. It is even possible, although unlikely to take O(1) time if the first try is the right answer.

It might have been tempting to try and optimize the solver by testing various starting positions and various start number, but this optimization would have only been valid for this particular puzzle. We felt that the best approach was to randomize the start location, as well as the start guess. In order to implement this, we used circular loops to scan the rows, columns and tested values. To keep track of how 'deep' we got in the algorithm, we incremented an index at every function call that made the function exit when it reached 81 (the maximum depth of the grid).

This approach also gives us great versatility with regards to multithreading environment. By starting at a random location, the solving function can be called by many threads and generate good 'far apart' starting states for each thread. This added feature gives the user to freely add threads without having to modify anything in the algorithm for the new threads to pick up a work load.

MULTI-THREADING AND SYNCHRONIZATION

Since the challenge is to test a very large possibility space, the use of parallel computing can greatly aid. For this particular problem, we used many threads to run a subset of the problem. Java was chosen as a platform because of its ease of implementation of multithreading environments. The main class takes care of creating threads, and every thread takes on a portion of the puzzle. As soon as one of them solves the problem, it prints out the solution and the time it took to solve it.

To handle synchronization, we used a global variable called 'finished'. When the main function starts, if sets 'finished' to false. As the threads try to solve the puzzle, they look at this variable and exit if it is true, otherwise continue to try and solve. As soon as a thread solves the puzzle, it sets the variable to true, which forces all the other threads out of their solving function. Meanwhile, in the main function, after all the threads have been initialized, the program falls into a while(!finished) loop. As soon as the barrier is crossed, we know that a thread has solved the puzzle, so all the threads are destroyed.

EXPERIMENT

Since, in general, we do not have *a priori* information on the best starting location for a brute force scan; we felt the need to test various scenarios to make sure that the data collected was a good reflection of reality. Therefore, in order to gather a comparative idea of various times to solve, rather than the particular value for this puzzle, many runs of the same test were performed. We also tested for many different numbers of threads.

The tests were conducted on an Inter Dual core processor @1.4GHz, managed by the Windows 7 platform.

RESULTS

First we ran some tests on a single thread to see how long it took on average to solve the puzzle. Here are the results we found.

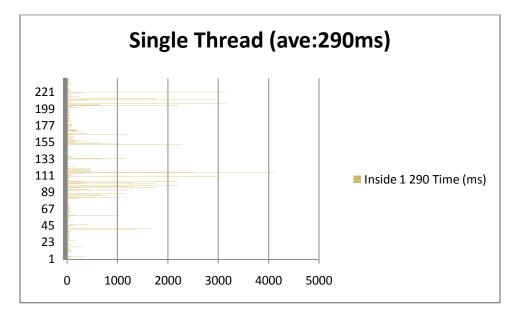


Figure 4: Individual results of a single threaded application solving the puzzle. On the x axis, we can see how long it took, and on the right, we can see the trial number.

These results justified the assumption made that the time it takes to fully solve a Sudoku puzzle on a brute force approach is greatly relative to the starting location. We can observe a few instances where it took 3-4 full seconds to solve, these represents where statistically we picked the worst possible starting point and the solution was one of the last ones we tested. As the other extreme, we can observe some instances where the puzzle was solved in a matter of milliseconds. These instances occur when the solution is one of the first ones to be found.

Afterwards, we ran the application with a varying number of threads working on solving the puzzle, from 1 to 10, and got the following results, which are similar to the original results we got. For any given number of threads, 500 tests were performed to ensure that the data gathered was statistically significant. (The number of different starting positions is 9x9 with 9 possible starting values, giving 729 possible initial states for the system.)

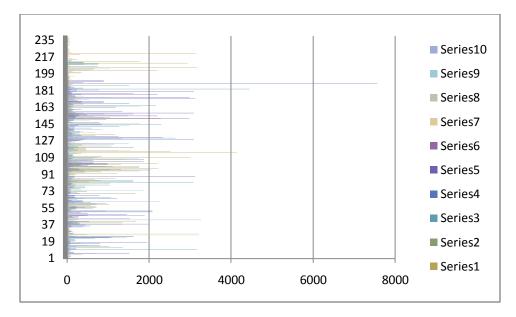


Figure 5: Combined results of all the tests performed with a varying number of threads. On the x axis, we can observe the time to solve, in ms.

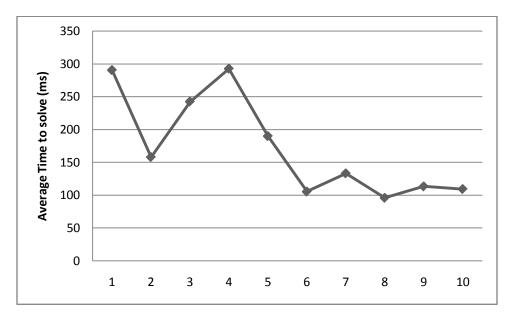


Figure 6: Average time to solve when using a variation of number of threads. On the x axis we can see the number of threads used, and on the y axis we can see the average time to solve in ms.

These are interesting results, which require a few explanations. First, as expected by running the code on a dual core processor, we can see that there is a sharp decrease in the average time required to solve the puzzle when going from a single threaded application to using 2 threads. This can intuitively be explained because of the fact that 2 processors can easily handle 2 threads without too much overhead.

When adding a 3rd and 4th thread, we can see that the average time to solve increased almost as far up as the original time it took for a single threaded application. This is caused by the added overhead brought by managing the extra threads without bringing too many benefits.

Afterwards, we observe a gradual decrease in the time to solve as the number of processors keeps increasing, until it bottoms out at 100ms. This continuous decrease is most probably due to the random nature of the starting position and starting number. As we can observe in figure 4, when comparing the performance we got from different starting locations, the data is mostly gathered in very low solving times (<200ms), with a few high peaks (>1s), which drive the average up. Since the vast majority of the starting positions can yield a fast answer on a brute force approach, by adding threads starting at random locations, we increase the chance that at least one of them starts in that location. If this is the case, as soon as this thread reaches the solution, all the other ones will be stopped anyways.

Finally, as the number of threads keeps increasing, we can observe a slight increase in the average time to solve. This average shows a weak asymptote at around 100ms, with a slight positive slope, indicating that the added threads do induce some extra overhead.

APPENDIX I: SOURCE CODE USED

The code consists of a single java file and was developed using the IDE Eclipse, and has been inspired by an algorithm proposed by Bob Carpenter on the following web page: http://www.colloquial.com/games/sudoku/java_sudoku.html.

package sudokuSolver; import java.util.Random;

// Class taking care of solving the Sudoku

public class solver extends Thread {

static boolean finished;

// This method is called when a thread starts running
 // It will call the recursive solver and output the data
 public void run() {
 int[]] board = setUpBoard(1);
 int i, j, value;

randomly selected

long start, end; Random generator = new Random(System.currentTimeMillis()); //printBoard(board);

> board = setUpBoard(1); i = generator.nextInt(8); j = generator.nextInt(8); value = generator.nextInt(8)+1;

System.out.println("No answers for this puzzle!");

end = System.currentTimeMillis();

if(!finished){

}

finished = true; System.out.print((end - start)+"\n");

}

public static void main(String[] args) {

// 1 threads thread System.out.println("\n\n1\nTime (ms)"); for(int k = 0; k < 400; k++){</pre>

> finished = false; Thread thread1 = new solver();

thread1.start(); while(!finished);

thread1.stop();

}

// 2 threads thread System.out.println("\n\n2\nTime (ms)"); for(int k = 0; k < 400; k++}{ finished = false; Thread thread1 = new solver(); Thread thread2 = new solver();

thread1.start();
thread2.start();

while(!finished);

// Set up the board (option to select different puzzles) // Start position (i,j) and start value will be thread1.stop(); thread1= null; thread2.stop(); thread2 = null;

}

// 3 threads thread System.out.println("\n\n3\nTime (ms)"); for(int k = 0; k < 400; k++){ finished = false; Thread thread1 = new solver(); Thread thread2 = new solver(); Thread thread3 = new solver();

> thread1.start(); thread2.start(); thread3.start();

while(!finished);

thread1.stop(); thread1= null; thread2.stop(); thread2 = null; thread3.stop(); thread3= null;

}

// 4 threads thread
System.out.println("\n\n4\nTime (ms)"); System.out.printin (\n\n\n\nime(ins)); for(int k = 0; k < 400; k++){ finished = false; Thread thread1 = new solver(); Thread thread2 = new solver(); Thread thread3 = new solver(); Thread thread4 = new solver();

> thread1.start(); thread2.start(); thread3.start(); thread4.start();

while(!finished);

thread1.stop(); thread1= null; thread2.stop(); thread2 = null; thread3.stop(); thread3= null; thread4.stop(); thread4 = null;

}

// 5 threads thread
System.out.println("\n\n5\nTime (ms)"); for(int k = 0; k < 400; k++){

finished = false; Thread thread1 = new solver(); Thread thread2 = new solver(); Thread thread3 = new solver(); Thread thread4 = new solver(); Thread thread5 = new solver();

thread1.start(); thread2.start(); thread3.start(); thread4.start(); thread5.start();

while(!finished);

thread1.stop(); thread1= null; thread2.stop(); thread2 = null; thread3.stop(); thread3= null; thread4.stop(); thread4 = null; thread5.stop(); thread5 = null;

}

thread1.start(); thread2.start(); thread3.start(); thread4.start(); thread5.start(); thread6.start();

while(!finished);

thread1.stop(); thread1= null; thread2.stop(); thread2 = null; thread3.stop(); thread3= null; thread4.stop(); thread4 = null; thread5 = null; thread6.stop(); thread6 = null;

}

// 7 threads thread System.out.println"(n\n7\nTime (ms)"); for{int k = 0; k < 400; k++}{ finished = false; Thread thread1 = new solver(); Thread thread2 = new solver(); Thread thread3 = new solver(); Thread thread4 = new solver(); Thread thread5 = new solver(); Thread thread5 = new solver(); Thread thread7 = new solver();

thread1.start();

thread2.start(); thread3.start(); thread4.start(); thread5.start(); thread6.start(); thread7.start();

while(!finished);

thread1.stop(); thread2=null; thread2.stop(); thread2.stop(); thread3.stop(); thread3.stop(); thread4.stop(); thread4.stop(); thread5 =null; thread6 =null; thread6 =null; thread6 =null;

}

thread1.start();

thread2.start(); thread3.start(); thread4.start(); thread5.start(); thread6.start(); thread7.start(); thread8.start();

while(!finished);

thread1.stop(); thread2=null; thread2.stop(); thread2 = null; thread3.stop(); thread4 = null; thread4.stop(); thread4 = null; thread5.stop(); thread6 = null; thread6 = null; thread7 = null; thread8.stop(); thread8 = null;

}

// 9threads thread
System.out.println("\n\n9\nTime (ms)");
for(int k = 0; k < 400; k++){
 finished = false;
 Thread thread1 = new solver();
 Thread thread2 = new solver();
 Thread thread3 = new solver();
 Thread thread4 = new solver();
 Thread thread5 = new solver();
 Thread thread5 = new solver();
 Thread thread6 = new solver();
 Thread thread7 = new solver();
 Thread thread9 = new solver();
 T

thread1.start(); thread2.start(); thread3.start(); thread4.start(); thread5.start(); thread6.start(); thread6.start(); thread8.start();

while(!finished);

thread1.stop(); thread2=null; thread2.stop(); thread2=null; thread3.stop(); thread3=null; thread4.stop(); thread5_stop(); thread5_stop(); thread5_stop(); thread6_stop(); thread6_stop(); thread7_null; thread8_stop(); thread8_stop(); thread8_stop(); thread8_null;

}

// 10 threads thread System.out.println("\n\n10\nTime (ms)"); for{int k = 0; k < 400; k++}{ finished = false; Thread thread1 = new solver(); Thread thread3 = new solver(); Thread thread3 = new solver(); Thread thread4 = new solver(); Thread thread5 = new solver(); Thread thread7 = new solver(); Thread thread8 = new solver(); Thread thread7 = new solver(); Thread thread8 = new solver(); Thread thread8 = new solver(); Thread thread7 = new solver(); Thread thread8 = new solver();

```
Thread thread9 = new solver();
Thread thread10 = new solver();
thread1.start();
thread2.start();
thread3.start();
thread4.start();
thread5.start();
thread6.start();
thread7.start();
thread8.start();
thread9.start();
thread10.start();
while(!finished);
thread1.stop();
thread1= null;
thread2.stop();
thread2 = null;
thread3.stop();
thread3= null;
thread4.stop();
thread4 = null;
thread5.stop();
thread5 = null;
thread6.stop();
thread6 = null;
thread7.stop();
thread7 = null;
thread8.stop();
thread8 = null;
thread9.stop();
thread9 = null;
thread10.stop();
thread10 = null;
```

}

}

static int[][] setUpBoard(int puzzleChoise) {

// Set up a new board, all formatted to 0 int[][] newBoard = new int[9][9]; // default 0 vals int i,j, n=0;

for(i = 0; i < 9; i++) for(j= 0; j < 9; j++)

newBoard[i][j] = puzzle[n++];

// printBoard(newBoard);

return newBoard;

}

static boolean solve(int row, int col, int]]] board,int xTimes, int startV) {
 // Return true if we have reached the depth of the recursion
 if(xTimes == 81) return true;
 if(finished) return true;

// Do a loop of rows and columns
if (++col == 9){
 col = 0;
 if(++row == 9)
 row = 0;

}

if (board[row][col] != 0){ // skip filled cells return solve(row ,col, board, xTimes+1, startV); }

// The code can only get here if the cell was 0 for (int val = 1; val <= 9; ++val) { if(++startV == 10) startV = 1;

```
// first check of the value is allowed here
if (allowedHere(row,col,startV,board)) {
    board[row][col] = startV; // If allowed, record it and run recursively
        if(solve(row ,col, board, XTimes+1, startV))
                                                                      return true;
        }
         board[row][col] = 0; // reset on backtrack
        return false;
     }
     static boolean allowedHere(int row, int col, int value, int[][] board) {
                        int i;
                        // scan 9 neighboring possibilities
for(i = 0; i < 9; i++){</pre>
                                              ; (++){
// look at colums in this row
if(board[row][i] == value)
return false;
// look at rows in this column
                                               if(board[i][col] == value)
                                              return fals;

// look at sub square

if (board[row/3*3+i/3][col/3*3+i/3] == value)
                                                                     return false;
                        }
        return true; // no violations, so it's legal
    }
     static void printBoard(int[][] boardToPrint) {
                        int i, j;
        for (i = 0; i < 9; i++) {
if( i%3 == 0)
               System.out.println(" -----
                                                                     -----");
           for (j = 0; j < 9; j++) {
    if (j%3 == 0) System.out.print(" | ");
    if(boardToPrint[i][j] == 0)
    System.out.print("* ");</pre>
                else
                        System.out.print(Integer.toString(boardToPrint[i][j])+ " ");
            }
            ,
System.out.println("|");
         }
         System.out.println(" -----");
}
}
```

APPENDIX II: RAW DATA

Here is the raw data gathered by the experiment. This first row, in bold, indicated the number of threads used in the test run. The second column is the average time it took to solve the puzzle whiles the experiment. The following columns are the individual time data harvested by repeatedly performing the test.

Note: due to the random nature of the data collected, repeating the experiment should yield a similar average, but different individual values.

290 153 242 190 105 133 96 113 195 1124 15 15 16 15 15 16	1	2	3	4	5	6	7	8	9	10
11241516161615151616161515169416151516151615157816941615151615161515375153116151516312116151515151516161617172161515151515151515151516783216151515151515151516783216151515151515151515171515151515151515151516 <th>290</th> <th>158</th> <th>242</th> <th>292</th> <th>190</th> <th>105</th> <th></th> <th>96</th> <th>113</th> <th>109</th>	290	158	242	292	190	105		96	113	109
15151515151516167816781615161516151678163112415117216311212161515151515151515151515161515615151515151515151516161516161616151615161616151615161516151617161615161615161516181371615161616161616181371515151616161616161813715151615161616161616191515161616161616161616161616161516	1124	15	31	265	15	15		16	16	16
78167816151615161316171516311241511716311617161515151515151515151516151515161516151515151616161616161516151615161616161616161516151615161516171611162115151515151515162048181371615151516 <td></td>										
175151515151515151721613167021161515151515161616161616167882161515151515151515461631721831161515153115317863172183115151516161531786315171615151616163131786315171615161678313178631517151616167831317863151615161616783131327878151615161616161631313278781516151616161631313231313231313231313231313231313231313231	78	16	94	16	15	16	47	31	16	16
15163116311631163116721615153116151515151515161516151615151515151546161615161516151515151515741615161516151615161516151678133151716151516<	78	16	78	16	15	15	16	15	16	15
11156215151516 <td>375</td> <td>15</td> <td>31</td> <td>16</td> <td>32</td> <td>16</td> <td>16</td> <td>31</td> <td>31</td> <td>19</td>	375	15	31	16	32	16	16	31	31	19
161515151515151516161616161516161516171616161615311531181371611161531161218137161116151616161618103015171615161616161617631517151615161616161618622278161516151616161616186217161616151616162031617161716	15	16	31	124	1515	172	16	31		702
167832161516151615161516151616171617183116151116181116181116181116181116181116 <td></td>										
46161616161616161631786317218313116161531203781301517163231163231318110301517161516169317476315171516151616161616128621219515151516161616161286218716151615161620331154763175161616151516203311516163105171616161515161615151676410716151616151516161515161051661616161616161616161610616161616161616161616110321516161616161616161611015161616161616161616161616161616161616 <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>										
471631672183116153131786317218313116161531317813571611163115151615164687815151616167831161516161674631578161516161616161615471516151616161513161547471871615161615131613233162156171616161515161314511511081071615151615151631311511081071615161515163131311511081071615151616313131151631151081071615151515163131311516313131151631 <td></td>										
7863172183131161616323131100015171632311632313147631517151516169317476315171516151616161616128622279311615161516161616161386217161516161616203203203203131621561716161616161620320316 </td <td></td>										
7813571561116323115312031311030151515151616781762481679515151616781713286232795311615151615161562781516161515151616157815161615151615161616161615151816151516151616161515161515161516161616161515161516161616161616161616153116<										
111030151716151516162504786263151715161616161616746315151515161516161616746315151516151615161616786278161616151516161620311563156781516161615151516167462670161030161515181616751681661663231161616311516317462670161515151631161616311631163116311631163116311631 <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>										
624816795151515169317378623279531161516141161547151950161516151615164747187161516311516327815474718716151631151520331627016161615327815317626701616161531151516616615151518151515151661661515151615321616171615321616153216311561661515161531161631161532151616161631311716151616161616313116153216161616161616161531161616161616161615311616161616161616153116161616 <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>										
47631517151616167831328623278195016151615161617627816151615161616161678627816151616161515162036315678151616161515203161561561716161615151715616161561515181611615151611716816167915161515313131167161615151511163131313131167161151516161631151631311718615151616161616311516311615151616161616151531311616301516 <td></td>										
1286232795311615161411615627816166215327815474718716161616152033162156171616161615203316217616161616152033162670161030161615151615616616616151516151616156166166161515161616164710632151616163131313131314715151411616161631<										
154715195016151615161516167862781516151631151620363156781516161616151615476267016103016151518145115511081077161516151518157115615616161616161632164710832153131321416314663321616162031631474843215313131153132931516311631313131314747161616164715323147161531311631323231471615313116323131324716163231311632313147161531311631323131471615313116311632314716153631161616<										
7862781616621532781547471871616161616203316278151616161520331626701610301615151515156110810771615161515151715616161616153232141621816474843215163232141616314715151411616203161631479316471251531151631471515161616101531316293161616101531314716153131153131314716153131153131471615313115163131471616390125154715151547161631311616161616174161631311616161616174161631313										
474718716151611151620363156781516161632311517626701610301615151814511511081077161515161515121615110820151615151615163147110321531313214151631471515141161602031631474843215313132141531473315161616203163152931516311516313162931516311515313162931516311515313162931531311515313162311632321647151531631516311619232471615161441515163116161616161615151515171515313131326<										
631567815161616161523316215617161030161515181451151081077161516151517161561616161616161615321647100321516323214151631471515141161401516313147151514116140151631313293151631311516313131313131316293151631161616313231										
1162156171616161612311518145115110810771615161515151615151615616616151615161516151615161771561611516321516321516313216163131153131311531313131311531										
47626701610301615151515151771561616161616167915161532164711032151632321415163147484321531313141516314715151411614015163132931516161631313115314747161616161616313131313131629315163115161616161616313131153147161531311516471515313131323132313132313131323131313132313131313131313231313131313131313131313131313131313132313131313131 </td <td></td>										
15108107161516161516151771561616161616791516153216164711032151632321416163147484321531313214151631471515141161401516313146633216471251531311516315293164712515313115163131629316153131151616313131313132316293161531311547151531313231623116311516471515213231631516444432281616151616156315164311615161616151616161541521516311615161616161616161641515163116151616161616161616										
15616116116179151615323216471103215313132141516314748432153116161631471515161616163146633216161631151631529315163116163115316293151631161401532314716161616140153231471616311516471515314716155619316471515124615164443228114032815161341152651251547151516134415265125151631161515416311613231215151615441515151631161532471615265125151611161515416311615161616161541616151616<										
471103215163231416162181647484321531313214151631471515141161602031616313293164712515311516316293164712515313115163162931616161401532314747161531311547161531623116301251547151511641516444322811403281516471615311631212151516154615164443228112151516154793265203163123121515161516134115151631140153264321615161451515163116151615161516154161516161616161616161616416151631163131 </td <td></td>										
47 15 15 14 16 10 15 16 16 31 46 63 32 16 16 16 203 16 16 31 52 93 15 16 31 15 16 31 62 93 15 16 31 15 31 31 15 31 62 93 15 16 16 140 15 31 31 47 16 15 31 15 47 16 15 31 62 31 16 30 125 15 47 15 15 21 47 16 15 561 93 16 47 31 16 132 46 15 16 444 32 281 140 328 15 16 47 93 265 203 16 312 312 15 15 16 1341 15 15 16 31 16 15 63 1683 16 134 16 15 16 13 16 15 154 16 <td>47</td> <td></td> <td></td> <td></td> <td>16</td> <td></td> <td>3214</td> <td>16</td> <td>218</td> <td></td>	47				16		3214	16	218	
46 63 32 16 16 16 203 16 16 31 32 93 16 47 125 15 31 15 16 31 47 47 16 16 11 16 31 31 15 31 47 16 15 31 31 15 47 16 15 31 62 31 16 31 15 16 47 15 15 31 47 16 15 561 93 16 47 31 16 132 46 15 15 15 16 132 15 16 1341 93 265 203 16 31 16 15 16 1341 15 265 125 15 16 31 140 15 264 15 16 31 16 15 <	47	484	32	15	31	31	3214	15	16	31
32 93 16 47 125 15 31 15 16 31 62 93 15 16 31 16 31 31 15 31 47 47 16 15 31 31 15 47 16 15 31 47 16 16 390 125 15 47 15 15 31 47 16 15 561 93 16 47 31 16 132 46 15 16 444 32 312 312 15 16 47 93 265 203 16 312 312 15 16 15 154 15 16 31 16 15 63 16 15 16 15 16 15 16 15 16 15 16 15 16 15 16 15 16 <	47	15	15	141	16	140	15	16	16	31
629315163116313115314747161531111516140153231471615313115164716153162311631151647151521471615561931647311613246151644432281140328151663151341931618232471615164152031631231215151613411526520316311616156155163116151631161516851624953131151516311615156163116151516313132321615152151515313132321616161616266313131323115161615500188816125161616151616161631313231313231161616	46	63	32	16	16	16	203	16	16	31
4747161616161401532314716153131154716153162311631151547151531471615561931647311613247161556193164731161324615164443228140328151663151341931619823247161564793265203163123121515161341152651251516311615631685162495313116311401532643228131151516311615161516111516311615161516161091528131163231161616161615161516151616161616313231125151516161616151615161616161616161616161616 <td>32</td> <td></td> <td>16</td> <td>47</td> <td>125</td> <td>15</td> <td></td> <td>15</td> <td>16</td> <td>31</td>	32		16	47	125	15		15	16	31
47161531311547161531623116311516471515314716155619316473116132461516444322811403281516631513419316122312471615641516444322813115161565151341931612231215151615416265203163123121515161685162495313116311615161551631151631311532643228131151516313132161515311516313132161616162663131313216161550018881612516161516151631313231311616164991631313231151616163116151516161616313132 <td>62</td> <td>93</td> <td>15</td> <td>16</td> <td>31</td> <td>16</td> <td>31</td> <td>31</td> <td>15</td> <td>31</td>	62	93	15	16	31	16	31	31	15	31
623116311516471515314716163901251547151521461516444322811403281516631513419316198232471615647932652031619823115151516134115265125151631161563168516249531311631161532632281311516311615326432281311515111615326432281311515311631163115163116153264323116313116322813115151531313132313132151616161616163131313231313231313231313231313231313231313231313231313231313231313231313131323131										
4716161639012515471515161324716155619316473116132461516444322814032815151663151341931619823247161516147932652031631231215151631168516249531311631161563168516249531311631140153264322813115151516311615165115215151516311616322813115151531163236161616266313116321516161616151615161516155001888161251515161516145132311251515161610207516151631313231151632761615161516151616161020751615 <td></td>										
47161556193164731161324615164443228114032815166315134193265203163123121515161341152651251516311615631685162495313116311615326415163116151631140153264322813115151631161532643228131151516311615326432281311515163116326435161515163131163264361616161631311632643715151515161616161516161626631313132161615161451323112515151616161449916313132311516161631327616313132311616161616161631327 <td></td>										
461516444322811403281516631513419316198232471615147932652031631231215151613411526512515163116156316851624953131163116161515163116151631106153264322813115152183116157815152915311516311615161616162663131313240615161550018881612516161516151645132313132311516151645132313132311516161451323131323131321616164715161632761615161631161632716163276161616781631161632761616311678161616 <td< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></td<>										
631513419316198232471615 47 932652031631231215151613411526512515163116156316851624953131163116151561516311615163116156151631151515218311532281311515152183116151561616161626631311632166161615500188816125151515161550018881612515151616164991631313231151616499163131323116161020751615163271515161631163271529313132311616311615161616313271529311516163116163132715293116161631161631161616 <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>										
4793265203163123121515161341152651251516311615631685162495313116311401532643228131151516314015326432281311515152183116311515291531151631163216161616266313131321616161615281153131321616161614513231125151516161649916313132311516161632716313271515151616102075161531327151516163132716153132715313271631167816311632761631167816311647153116311615161531161631167815166241531161616<										
13411526512515163116156316851624953131163116151561516311615163114015326432281311515152183131157815152915311516311616151616161626631313132406151615500188816125161615161516445132313132311516161646313132311515161616327163131323115151516161020751615163276163132731323132313231323131323131 <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>										
16851624953131163116161515163116151631140153264322813115151521831311578151529153115163116151561616162663131163216161615500188816125161615161550018881612515151616151614513231313231151616164991631313231161611020751616313271529311616311631164715311616311631327616311678163116327616312037815166241593109151637181.93E+022343142116471516164781615163171615657716151516317161647816151516 <td></td>										
151631161516311401532643228131151515218313115781515291531151631163215616161616266313131324061516155001888161251515151615164451323131323115151632716313132311516162075161516327151631323115151632716313271521515161632716151632752151516163271615163275231323116362075161516327533132311631781516161631323132311631781516624159310915163120378151516317161631161615151616151516163116 </td <td></td>										
32281311515152183131157815152915311516311615156161616162663131163216161610915281153131324061516155001888161251616151615164991631313231161616327163131323115161610207516151632753161616327161516327533132731161630620751615163276161401693620751615163276161401678163115163131321631472181710916218156163123151516641593109151631193E+0223431421164715161615151616151516163116161616151516<										
781515291531151631161515616161616266313131321616109152811531313240615161550018881612516161516151614513231125151516161649916313132311516163271616313271529311616100207516163132761631167816311614015313114016936207516151621815616313178151662415931091563718193E+022343142116471516156577151515161516317161029718151616151516163131262163112537763161615151626216311253776316161631313216161515<										
15616161616266313116321616161091528115313132406151615500188816125161615161516145132312251515161516164991631313231151516327163131323116162075161631327152931140169362075161516327616311678163116140153116314721817109162181601631203781516624159310915161631193E+02234314211647151616151515151516317161615161616257716161515161631161615161631151616477879511616151516163115161615161616151616<										
406151615500188816125161615161516145132311251515161616499163131323115151632716313147151516162075161631327152931140169362075161516327616311678163116140153116314721817109162181561631203781516624159310915637181.93E+022343142116471516156577151515219151091516102971815161515161631164768795116161515161631154716391343161615151622621631125377631616163132109163216151516163132161615161615161616 <td>156</td> <td>16</td> <td>16</td> <td>16</td> <td>16</td> <td>266</td> <td>31</td> <td>31</td> <td>16</td> <td></td>	156	16	16	16	16	266	31	31	16	
15161516145132311251515161616499163131323115151632716313132311616110207516163132715293114016936207516151632761631167816311614015311631781516624159310915631203781516624159310915633718193E+022343142116471516156577151515219151091516156577161615163171610297181516151516163131164768795116161515161631321647163913431616151516226216311253776316163131321991632161515161631321616151516163132	16	16	16	109	15	281	15	31	31	32
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	406	15	16	15	500	1888	16	125	16	16
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	15	16	15	16	1451	32	31	125	15	15
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	16	16	16	16	499	16	31			31
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$										
1029 718 15 16 15 47 16 15 31 16 47 687 951 16 16 15 15 16 16 31 47 577 391 187 15 171 16 15 31 15 47 16 391 343 16 16 15 16 2262 16 31 125 377 63 16 16 16 31 15 109 16 32 16 15 16 16 31 32 47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16 16 16										
47 687 951 16 16 15 15 16 16 31 47 577 391 187 15 171 16 15 31 15 47 16 391 343 16 16 15 16 262 16 31 125 377 63 16 16 31 15 109 16 32 16 15 16 16 31 32 47 15 250 1217 421 141 16 16 31 32 109 16 32 16 15 15 16 13 32 47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16										
47 577 391 187 15 171 16 15 31 15 47 16 391 343 16 16 15 15 16 2262 16 31 125 377 63 16 16 16 31 15 109 16 32 16 15 15 16 31 32 47 15 250 1217 421 141 16 16 31 32 46 16 31 2 203 15 16 15 16 16										
47 16 391 343 16 16 15 15 16 2262 16 31 125 377 63 16 16 16 31 15 109 16 32 16 15 15 16 16 31 32 47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16										
16 31 125 377 63 16 16 16 31 15 109 16 32 16 15 15 16 16 31 32 47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16										
109 16 32 16 15 15 16 16 31 32 47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16 16										
47 15 250 1217 421 141 16 16 172 16 46 16 31 2 203 15 16 15 16 16										
46 16 31 2 203 15 16 15 16 16										
47 15 1061 64 171 16 62 16 109 15	47	15	1061	64			62	16		

63	32	15	15	171	16	31	15	203	15
62	15	15	172	780	16	16	15	93	16
47	16	16	15	16	47	16	16	16	16
47	1653	78	312	62	31	15	16	31	16
62	250	78	94	62	31	15	15	31	15
16	484	78	16	47	78	32	15	433	32
31	16	16	94	16	78	343	15	1872	17
15	31	78	93	31	31	359	16	2	15
33	16	297	78	31	16	78	16	16	16
16	16	436	163	109	124	78	31	15	15
15	16	436	78	156	31	172	16	16	16
15	15	187	32	109	47	203	15	16	436
15	16	1029	78	78	16	561	16	16	16
203	78	125	78	16	125	156	15	31	15
1045	15	125	78	15	16	31	16	16	31
546	16	3074	93	16	93	514	16	16	16
47	16	1607	31	31	234	141	15	15	15
811	15	1607	93	16	16	546	16	16	31
671	16	140	31	15	16	16	15	16	312
1170	15	32	109	32	15	16	16	16	94
16	31	188	31	15	16	172	16	1	140
32	16	171	62	3120	15	16	15	15	31
16	31						10		
		63	15	31	15	171		16	31
218	109	62	24	47	16	171	10	15	31
1217	171	31	33	16	16	16	15	15	16
438	33	31	33	16	16	16	16	16	31
31	811	109	109	15	62	827	452	15	16
1763	62	109	15	16	406	16	234	16	16
281	1045	94	2028	15	31	16	203	15	15
1311	281	312	16	187	78	15	608	32	15
2215	281	140	58	16	47	15	16	15	15
31	1747	16	47	16	78	31	15	16	16
1763	1108	936	32	156	31	16	31	16	15
281	343	94	32	499	125	16	15	15	15
1311	343	31	31	593	982	15	16	16	31
2215	125	109	640	16	982	16	16	31	16
343	343	16	640	63	15	32	16	46	31
47	125	16	1872	16	32	234	15	31	16
47	125	468	640	16	15	561	15	16	31
47	16	125	1872	32	31	1420	31	31	31
62	15	78	125	15	31	16	1731	16	16
47	16	624	73	16	16	718	733	31	15
2995	109	187	15	16	16	110	62	31	62
63	173	844	16	16	125	15	15	16	31
125	125	156	15	16	31	16	31	15	16
31	125	140	16	16	31	31	31	219	32
343	16	16	31	47	31	15	15	15	15
4134	16	16	31	15	16	16	453	218	15
2512	15	94	16	16	110	281	172	47	16
460	16	250	15	15	78	1529	1248	203	16
460	15	250	16	16	47	15	31	158	16
460	47	16	160	31	16	15	31	31	15
460	281	172	1616	15	47	15	17	31	16
		47							
46	265		15	16	16	15	171	249	16
16	1077	48	15	15	16	437	359	32	15
156	142	31	16	16	16	15	359	31	16
15	1139	78	16	16	15	296	670	15	1498
16	94	62	15	16	16	16	31	32	193
15	15	32	15	15	15	219	15	31	31
32	31	109	16	32	280	15	15	15	31
15	78	16	327	16	406		15		171
						16		32	
16	140	46	3089	31	15	234	16	2636	468
15	16	110	1332	15	31	16	15	2636	16
16	32	297	16	2325	16			889	16
16						47	16		
	141	109	31	1248	15	47 156	16 32	31	16
686	141 16								
	16	109 109	31 343	1248 16	15 16	156 16	32 32	31	16 16
1170	16 16	109 109 312	31 343 16	1248 16 16	15 16 109	156 16 47	32 32 16	31 156	16 16 15
1170 31	16 16 78	109 109 312 109	31 343 16 15	1248 16 16 15	15 16 109 187	156 16 47 16	32 32 16 31	31 156 16	16 16 15 296
1170 31 125	16 16 78 47	109 109 312 109 109	31 343 16 15 32	1248 16 16 15 16	15 16 109 187 32	156 16 47 16 31	32 32 16 31 687	31 156 16 78	16 16 15 296 296
1170 31 125 46	16 16 78 47 16	109 109 312 109 109 312	31 343 16 15 32 15	1248 16 16 15 16 31	15 16 109 187 32 31	156 16 47 16 31 16	32 32 16 31 687 32	31 156 16 78 390	16 16 15 296 296 31
1170 31 125	16 16 78 47	109 109 312 109 109	31 343 16 15 32	1248 16 16 15 16	15 16 109 187 32	156 16 47 16 31	32 32 16 31 687	31 156 16 78	16 16 15 296 296
1170 31 125 46	16 16 78 47 16 78	109 109 312 109 109 312 172	31 343 16 15 32 15	1248 16 16 15 16 31	15 16 109 187 32 31	156 16 47 16 31 16 15	32 32 16 31 687 32 172	31 156 16 78 390 16	16 16 15 296 296 31
1170 31 125 46 16 16	16 16 78 47 16 78 47	109 109 312 109 109 312 172 172	31 343 16 15 32 15 15 15	1248 16 15 16 31 64 94	15 16 109 187 32 31 31 141	156 16 47 16 31 16 15 47	32 32 16 31 687 32 172 78	31 156 16 78 390 16 16	16 16 15 296 296 31 164 842
1170 31 125 46 16 16 15	16 16 78 47 16 78 47 15	109 109 312 109 312 109 312 172 172 172	31 343 16 15 32 15 15 16 62	1248 16 15 16 31 64 94 140	15 16 109 187 32 31 31 141 187	156 16 47 16 31 16 15 47 173	32 32 16 31 687 32 172 78 31	31 156 16 78 390 16 16 15	16 16 15 296 31 164 842 15
1170 31 125 46 16 16 15 16	16 16 78 47 16 78 47 15 63	109 109 312 109 312 172 172 172 172 592	31 343 16 15 32 15 15 16 62 62	1248 16 15 16 31 64 94 140 15	15 16 109 187 32 31 31 141 187 31	156 16 47 16 31 16 15 47 173 31	32 32 16 31 687 32 172 78 31 16	31 156 16 78 390 16 16 15 15	16 16 15 296 296 31 164 842 15 33
1170 31 125 46 16 16 15 16 15	16 16 78 47 16 78 47 15 63 15	109 109 312 109 312 172 172 172 172 592 187	31 343 16 15 32 15 15 16 62 62 16	1248 16 15 16 31 64 94 140 15 15	15 16 109 187 32 31 31 141 187 31 47	156 16 47 16 31 16 15 47 173 31 15	32 32 16 31 687 32 172 78 31 16 16	31 156 16 78 390 16 16 15 15 16 11	16 16 15 296 31 164 842 15 33 15
1170 31 125 46 16 16 15 16 15 16	16 16 78 47 16 78 47 15 63 15 47	109 109 312 109 312 172 172 172 172 592 187 1264	31 343 16 15 32 15 15 16 62 62 16 15	1248 16 16 31 64 94 140 15 15 15	15 16 109 187 32 31 31 141 187 31 47 31	156 16 47 16 31 15 47 173 31 15 94	32 32 16 31 687 32 172 78 31 16 16 31	31 156 16 78 390 16 16 15 16 11 1515	16 16 15 296 296 31 164 842 15 33 15 15
1170 31 125 46 16 16 15 16 15	16 16 78 47 16 78 47 15 63 15	109 109 312 109 312 172 172 172 172 592 187	31 343 16 15 32 15 15 16 62 62 16	1248 16 15 16 31 64 94 140 15 15	15 16 109 187 32 31 31 141 187 31 47	156 16 47 16 31 16 15 47 173 31 15	32 32 16 31 687 32 172 78 31 16 16	31 156 16 78 390 16 16 15 15 16 11	16 16 15 296 31 164 842 15 33 15
1170 31 125 46 16 16 15 16 15 16	16 16 78 47 16 78 47 15 63 15 47	109 109 312 109 312 172 172 172 172 592 187 1264	31 343 16 15 32 15 15 16 62 62 16 15	1248 16 16 31 64 94 140 15 15 15	15 16 109 187 32 31 31 141 187 31 47 31	156 16 47 16 31 15 47 173 31 15 94	32 32 16 31 687 32 172 78 31 16 16 31	31 156 16 78 390 16 16 15 16 11 1515	16 16 15 296 296 31 164 842 15 33 15 15
1170 31 125 46 16 15 15 16 15 16 15 16 16 16	16 16 78 47 16 78 47 15 63 15 47 63 232	109 109 312 109 312 172 172 172 172 592 187 1264 94 2293	31 343 16 15 32 15 15 16 62 62 16 15 16 16	1248 16 15 16 31 64 94 140 15 15 15 265 844	15 16 109 187 32 31 31 141 187 31 47 31 47 31	156 16 47 16 31 16 15 47 173 31 15 94 62 94	32 32 16 31 687 32 172 78 31 16 16 16 31 15 15	31 156 16 78 390 16 15 16 11 1515 16 16 16	16 16 296 296 31 164 842 15 33 15 15 16 16
1170 31 125 46 16 16 15 16 15 16 15 16 15 16	16 16 78 47 16 78 47 15 63 15 63 15 47 62 32 31	109 109 312 109 109 312 172 172 172 172 172 172 187 1264 94 2293 16	31 343 16 15 32 15 16 62 62 16 15 16 16 16 16	1248 16 15 16 31 64 94 140 15 15 15 156 265 844 125	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828	156 16 47 16 31 15 47 173 31 15 94 62 94 62	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763	31 156 16 78 390 16 15 16 11 1515 16 16 15	16 15 296 296 31 164 842 15 33 15 15 15 16 16 16
1170 31 125 46 16 15 16 15 16 16 15 16 15 16 15 16 15	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31	109 109 312 109 312 172 172 172 172 592 187 1264 94 2293 16 125	31 343 16 15 32 15 16 62 62 16 15 16 16 16 16 16 15	1248 16 15 16 31 64 94 140 15 156 265 844 125 780	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828 16	156 16 47 16 31 16 15 47 173 31 15 94 62 94 62 94	32 32 16 31 687 32 172 78 31 16 16 31 15 15 15 1763 15	31 156 16 78 390 16 15 15 16 11 1515 16 16 15 15 16	16 16 296 296 31 164 842 15 33 15 15 15 16 16 16 16 47
1170 31 125 46 16 15 15 16 15 16 15 16 15 16 15 17	16 16 78 47 16 78 47 15 63 15 63 15 47 62 32 31 31 31 47	109 109 312 109 312 172 172 172 592 187 1264 94 2293 16 125 78	31 343 16 15 32 15 15 16 62 62 16 15 16 16 16 16 16 15 15 15	1248 16 15 15 64 94 140 15 15 156 265 844 125 780 31	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828 16 31	156 16 47 16 31 16 15 47 173 31 15 94 62 94 62 94 62 94 62 94 62	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763 15 31	31 156 16 78 390 16 15 15 15 16 16 16 15 16 15 16 15	16 16 296 296 31 164 842 15 33 15 15 16 16 16 16 47 15
1170 31 125 46 16 15 16 15 16 15 16 15 16 15 16 15 16 15 17 31	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31 31 47 46	109 109 312 109 109 312 172 172 172 172 172 187 1264 94 2293 16 125 78 140	31 343 16 15 32 15 15 16 62 62 16 15 16 16 16 16 16 15 15 47	1248 16 15 15 64 94 140 15 15 15 265 844 125 780 31 140	15 16 109 187 32 31 141 187 31 47 31 47 31 828 16 31 31	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 94 62 546	32 32 16 31 687 32 172 78 31 16 16 16 31 15 15 15 15 15 31 15 31 172	31 156 16 78 390 16 15 15 16 11 1515 16 15 15 16 15 31	16 16 296 296 31 164 842 15 33 15 15 16 16 16 16 47 15 16
1170 31 125 46 16 15 15 16 15 16 15 16 15 16 15 17	16 16 78 47 16 78 47 15 63 15 63 15 47 62 32 31 31 31 47	109 109 312 109 312 172 172 172 592 187 1264 94 2293 16 125 78	31 343 16 15 32 15 15 16 62 62 16 15 16 16 16 16 16 15 15 15	1248 16 15 15 64 94 140 15 15 156 265 844 125 780 31	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828 16 31	156 16 47 16 31 16 15 47 173 31 15 94 62 94 62 94 62 94 62 94 62	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763 15 31	31 156 16 78 390 16 15 15 15 16 16 16 15 16 15 16 15	16 16 296 296 31 164 842 15 33 15 15 16 16 16 16 47 15
1170 31 125 46 16 15 16 15 16 15 16 15 16 15 16 15 16 15 17 31	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31 31 47 46	109 109 312 109 109 312 172 172 172 172 172 187 1264 94 2293 16 125 78 140	31 343 16 15 32 15 15 16 62 62 16 15 16 16 16 16 16 15 15 47	1248 16 15 15 64 94 140 15 15 15 265 844 125 780 31 140	15 16 109 187 32 31 141 187 31 47 31 47 31 828 16 31 31	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 94 62 546	32 32 16 31 687 32 172 78 31 16 16 16 31 15 15 15 15 15 31 15 31 172	31 156 16 78 390 16 15 15 16 11 1515 16 15 15 16 15 31	16 16 296 296 31 164 842 15 33 15 15 16 16 16 16 47 15 16
1170 31 125 46 16 15 15 16 15 16 15 16 15 15 17 31 16 15 15	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31 47 46 47 31	109 109 312 109 312 172 172 172 592 187 1264 94 2293 16 125 78 140 951 171	31 343 16 15 32 15 16 62 62 62 16 15 16 16 15 15 15 47 16 16 16	1248 16 15 16 31 64 94 140 15 156 265 844 125 780 31 140 156 2980	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828 16 31 31 31 31	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 546 32 16	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763 15 31 172 171 16	31 156 16 78 390 16 15 16 15 16 15 16 15 16 15 31 15 562	16 16 296 296 31 164 842 15 33 15 15 16 16 16 47 15 16 47 421
1170 31 125 46 16 15 16 15 16 15 16 15 16 15 17 31 16 15 2278	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31 47 46 47 31 47	109 109 312 109 109 312 172 172 172 172 592 187 1264 94 2293 16 125 78 140 951 171 140	31 343 16 15 32 15 15 16 62 16 16 16 16 16 16 16 15 15 47 16 16 15 15	1248 16 15 15 64 94 140 15 15 15 265 844 125 780 31 140 156 2980 187	15 16 109 187 32 31 141 187 31 47 31 47 31 828 16 31 31 31 31 31 31 203	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 546 32 16 15	32 32 16 31 687 32 172 78 31 16 16 31 15 15 15 15 15 31 172 171 16 31	31 156 16 78 390 16 15 15 16 11 1515 16 15 16 15 31 15 31 15 62 15	16 16 296 296 31 164 842 15 33 15 15 16 16 16 16 16 47 15 16 47 421 905
1170 31 125 46 16 15 16 15 16 15 16 15 16 15 17 31 16 15 2278 2278 1045	16 16 78 47 16 78 47 15 63 15 47 62 31 31 31 47 46 47 31 47 47	109 109 312 109 109 312 172 172 172 172 172 172 172 172 187 1264 94 2293 16 125 78 140 951 171 140 219	31 343 16 15 32 15 16 62 62 62 16 16 16 16 16 16 15 5 47 16 16 15 32	1248 16 15 15 64 94 140 15 15 15 156 265 844 125 780 31 140 156 2980 187 187	15 16 109 187 32 31 141 187 31 47 31 47 31 828 16 31 31 31 31 31 31 203 78	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 546 32 16 15 16	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763 15 31 172 171 16 31 64	31 156 16 78 390 16 15 15 16 11 1515 16 15 15 16 15 31 15 52 15 16	16 15 296 296 31 164 842 15 33 15 15 16 16 47 15 16 47 421 905 218
1170 31 125 46 16 15 15 16 15 16 15 16 15 15 17 31 16 15 2278 1045 655	16 16 78 47 16 78 47 15 63 15 47 62 32 31 31 47 46 47 31 47 47 297	109 109 312 109 109 312 172 172 172 172 172 187 1264 94 2293 16 125 78 140 951 171 140 219 93	31 343 16 15 32 15 16 62 62 62 16 15 16 16 16 15 15 15 47 16 16 16 15 32 32 1467	1248 16 15 16 31 64 94 140 15 15 156 265 844 125 780 31 140 156 2980 187 187 2200	15 16 109 187 32 31 141 187 31 47 31 47 31 47 31 828 16 31 31 31 31 31 31 31 203 78 16	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 94 62 546 32 16 15 16 15	32 32 16 31 687 32 172 78 31 16 16 31 15 15 15 15 15 31 1763 15 31 1772 171 16 31 45 54 15	31 156 16 78 390 16 15 16 15 16 15 16 15 16 15 31 15 62 15 62 15 16 15	16 16 296 296 31 164 842 15 33 15 15 16 16 47 15 16 47 421 905 218 16
1170 31 125 46 16 15 16 15 16 15 16 15 16 15 17 31 16 15 2278 2278 1045	16 16 78 47 16 78 47 15 63 15 47 62 31 31 31 47 46 47 31 47 47	109 109 312 109 109 312 172 172 172 172 172 172 172 172 187 1264 94 2293 16 125 78 140 951 171 140 219	31 343 16 15 32 15 16 62 62 62 16 16 16 16 16 16 15 5 47 16 16 15 32	1248 16 15 15 64 94 140 15 15 15 156 265 844 125 780 31 140 156 2980 187 187	15 16 109 187 32 31 141 187 31 47 31 47 31 828 16 31 31 31 31 31 31 203 78	156 16 47 16 31 15 47 173 31 15 94 62 94 62 94 62 94 62 546 32 16 15 16	32 32 16 31 687 32 172 78 31 16 16 31 15 15 1763 15 31 172 171 16 31 64	31 156 16 78 390 16 15 16 11 1515 16 15 15 16 15 31 15 52 15 16	16 15 296 296 31 164 842 15 33 15 15 16 16 47 15 16 47 421 905 218

187	280	79	904	125	1616	78	156	15	16
375	16	78	170	3089	78	16	406	15	31
125	16	94	203	47	78	156	15	15	15
124	31	78	1341	16	31	32	16	16	16
78	16	140	31	16	111	16	16	15	16
125	171	31	345	16	16	78	15	16	47
125	31	93	16	15	16	47	16	15	31
16	47	93	1186	32	16	47	16	15	16
78	64	156	15	31	16	125	1763	16	15
1185	46	2168	15	16	15	109	15	16	15
124	47	48	16	156	16	31	31	47	16
390	62	62	1515	390	16	31	172	31	16
15	78	62	16	889	15	187	171	16	63
327	47	32	124	889	16	16	16	15	31
171	62	31	359	16	31	31	31	16	15
203	47	15	254	31	31	15	64	530	16
187	31	16	125	140	31	327	15	196	31
47	16	16	3123	156	31	374	16	1045	31
47	171	15	15	2980	203	16	156	16	218
47	48	16	15	187	78	78	406	16	62
111	47	16	31	187	16	16	15	16	15
93	16	16	343	2200	31	31	16	15	178
111	15	32	125	31	1616	32	16	16	79
93	16	32	16	125	78	16	15	15	31
16	15	32	15	3089	78	15	16	31	203
15	15	15	231	47	31	16	16	15	15