**8. The four adjacent digits in the 1000-digit number that have the greatest product are**

**9 × 9 × 8 × 9 = 5832.**

73167176531330624919225119674426574742355349194934

96983520312774506326239578318016984801869478851843

85861560789112949495459501737958331952853208805511

12540698747158523863050715693290963295227443043557

66896648950445244523161731856403098711121722383113

62229893423380308135336276614282806444486645238749

30358907296290491560440772390713810515859307960866

70172427121883998797908792274921901699720888093776

65727333001053367881220235421809751254540594752243

52584907711670556013604839586446706324415722155397

53697817977846174064955149290862569321978468622482

83972241375657056057490261407972968652414535100474

82166370484403199890008895243450658541227588666881

16427171479924442928230863465674813919123162824586

17866458359124566529476545682848912883142607690042

24219022671055626321111109370544217506941658960408

07198403850962455444362981230987879927244284909188

84580156166097919133875499200524063689912560717606

05886116467109405077541002256983155200055935729725

71636269561882670428252483600823257530420752963450

Find the thirteen adjacent digits in the 1000-digit number that have the greatest

product. What is the value of this product?

Ans: 23514624000

public class AdjacentDigits {

public static void main(String[] args) {

int adj = 13; // digits

String num = "7316717653133062491922511967442657474235534919493496983520312774506326239578318016984801869478851843858615607891129494954595017379583319528532088055111254069874715852386305071569329096329522744304355766896648950445244523161731856403098711121722383113622298934233803081353362766142828064444866452387493035890729629049156044077239071381051585930796086670172427121883998797908792274921901699720888093776657273330010533678812202354218097512545405947522435258490771167055601360483958644670632441572215539753697817977846174064955149290862569321978468622482839722413756570560574902614079729686524145351004748216637048440319989000889524345065854122758866688116427171479924442928230863465674813919123162824586178664583591245665294765456828489128831426076900422421902267105562632111110937054421750694165896040807198403850962455444362981230987879927244284909188845801561660979191338754992005240636899125607176060588611646710940507754100225698315520005593572972571636269561882670428252483600823257530420752963450";

long max=0;

for(int i=0; i<num.length()-adj; i++)

{

long x = 1;

for(int j=i; j<i+adj; j++)

{

int k = num.charAt(j)-'0';

x\*=k;

}

if(max<x) max = x;

}

System.out.println(max);

}

}

**9. A Pythagorean triplet is a set of three natural numbers, a < b < c, for which, a2 + b2 = c2**

**For example, 32 + 42 = 9 + 16 = 25 = 52.**

**There exists exactly one Pythagorean triplet for which a + b + c = 1000.**

**Find the product abc.**

**Ans: 31875000**

public class Triplet {

public static void main(String[] args) {

int num = 1000;

long ans;

long product = -1;

for(int a = 1; a<num/3; a++){

int b = (num\*num-2\*a\*num)/(2\*num-2\*a);

int c = num-a-b;

if(c\*c==(a\*a+b\*b)){

ans = a\*b\*c;

if(ans>product){

product = ans;

}

System.out.println(product);

}

}

}

}

**12. The sequence of triangle numbers is generated by adding the natural numbers. So**

**the 7th triangle number would be 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. The first**

**ten terms would be:**

**1, 3, 6, 10, 15, 21, 28, 36, 45, 55, ...**

**Let us list the factors of the first seven triangle numbers:**

**1: 1**

**3: 1,3**

**6: 1,2,3,6**

**10: 1,2,5,10**

**15: 1,3,5,15**

**21: 1,3,7,21**

**28: 1,2,4,7,14,28**

**We can see that 28 is the first triangle number to have over five divisors.**

**What is the value of the first triangle number to have over five hundred divisors?**

**Ans: 76576500**

public class TriangleNum {

public static int triangleNum(int n) {

int sum=0;

for (int i = 0; i <=n ; i++) {

sum+=i;

}

return sum;

}

public static void main(String[] args) {

int divisors = 0;

int size = 500;

int num = 0;

int j = 0;

while(divisors <= size) {

divisors = 0;

j++;

num = triangleNum(j);

for (int i = 1; i <= Math.sqrt(num); i++){

if (num % i == 0) divisors++;

}

divisors\*=2;

}

System.out.println(num);

}

}

**13.Work out the first ten digits of the sum of the following one-hundred 50-digit numbers.**

import java.math.BigInteger;

public class SumofTen {

private static String[] NUMBERS = {

"37107287533902102798797998220837590246510135740250",

"46376937677490009712648124896970078050417018260538",

"74324986199524741059474233309513058123726617309629",

"91942213363574161572522430563301811072406154908250",

"23067588207539346171171980310421047513778063246676",

"89261670696623633820136378418383684178734361726757",

"28112879812849979408065481931592621691275889832738",

"44274228917432520321923589422876796487670272189318",

"47451445736001306439091167216856844588711603153276",

"70386486105843025439939619828917593665686757934951",

"62176457141856560629502157223196586755079324193331",

"64906352462741904929101432445813822663347944758178",

"92575867718337217661963751590579239728245598838407",

"58203565325359399008402633568948830189458628227828",

"80181199384826282014278194139940567587151170094390",

"35398664372827112653829987240784473053190104293586",

"86515506006295864861532075273371959191420517255829",

"71693888707715466499115593487603532921714970056938",

"54370070576826684624621495650076471787294438377604",

"53282654108756828443191190634694037855217779295145",

"36123272525000296071075082563815656710885258350721",

"45876576172410976447339110607218265236877223636045",

"17423706905851860660448207621209813287860733969412",

"81142660418086830619328460811191061556940512689692",

"51934325451728388641918047049293215058642563049483",

"62467221648435076201727918039944693004732956340691",

"15732444386908125794514089057706229429197107928209",

"55037687525678773091862540744969844508330393682126",

"18336384825330154686196124348767681297534375946515",

"80386287592878490201521685554828717201219257766954",

"78182833757993103614740356856449095527097864797581",

"16726320100436897842553539920931837441497806860984",

"48403098129077791799088218795327364475675590848030",

"87086987551392711854517078544161852424320693150332",

"59959406895756536782107074926966537676326235447210",

"69793950679652694742597709739166693763042633987085",

"41052684708299085211399427365734116182760315001271",

"65378607361501080857009149939512557028198746004375",

"35829035317434717326932123578154982629742552737307",

"94953759765105305946966067683156574377167401875275",

"88902802571733229619176668713819931811048770190271",

"25267680276078003013678680992525463401061632866526",

"36270218540497705585629946580636237993140746255962",

"24074486908231174977792365466257246923322810917141",

"91430288197103288597806669760892938638285025333403",

"34413065578016127815921815005561868836468420090470",

"23053081172816430487623791969842487255036638784583",

"11487696932154902810424020138335124462181441773470",

"63783299490636259666498587618221225225512486764533",

"67720186971698544312419572409913959008952310058822",

"95548255300263520781532296796249481641953868218774",

"76085327132285723110424803456124867697064507995236",

"37774242535411291684276865538926205024910326572967",

"23701913275725675285653248258265463092207058596522",

"29798860272258331913126375147341994889534765745501",

"18495701454879288984856827726077713721403798879715",

"38298203783031473527721580348144513491373226651381",

"34829543829199918180278916522431027392251122869539",

"40957953066405232632538044100059654939159879593635",

"29746152185502371307642255121183693803580388584903",

"41698116222072977186158236678424689157993532961922",

"62467957194401269043877107275048102390895523597457",

"23189706772547915061505504953922979530901129967519",

"86188088225875314529584099251203829009407770775672",

"11306739708304724483816533873502340845647058077308",

"82959174767140363198008187129011875491310547126581",

"97623331044818386269515456334926366572897563400500",

"42846280183517070527831839425882145521227251250327",

"55121603546981200581762165212827652751691296897789",

"32238195734329339946437501907836945765883352399886",

"75506164965184775180738168837861091527357929701337",

"62177842752192623401942399639168044983993173312731",

"32924185707147349566916674687634660915035914677504",

"99518671430235219628894890102423325116913619626622",

"73267460800591547471830798392868535206946944540724",

"76841822524674417161514036427982273348055556214818",

"97142617910342598647204516893989422179826088076852",

"87783646182799346313767754307809363333018982642090",

"10848802521674670883215120185883543223812876952786",

"71329612474782464538636993009049310363619763878039",

"62184073572399794223406235393808339651327408011116",

"66627891981488087797941876876144230030984490851411",

"60661826293682836764744779239180335110989069790714",

"85786944089552990653640447425576083659976645795096",

"66024396409905389607120198219976047599490197230297",

"64913982680032973156037120041377903785566085089252",

"16730939319872750275468906903707539413042652315011",

"94809377245048795150954100921645863754710598436791",

"78639167021187492431995700641917969777599028300699",

"15368713711936614952811305876380278410754449733078",

"40789923115535562561142322423255033685442488917353",

"44889911501440648020369068063960672322193204149535",

"41503128880339536053299340368006977710650566631954",

"81234880673210146739058568557934581403627822703280",

"82616570773948327592232845941706525094512325230608",

"22918802058777319719839450180888072429661980811197",

"77158542502016545090413245809786882778948721859617",

"72107838435069186155435662884062257473692284509516",

"20849603980134001723930671666823555245252804609722",

"53503534226472524250874054075591789781264330331690",

};

public static String sum() {

BigInteger sum = BigInteger.ZERO;

for (String num : NUMBERS)

sum = sum.add(new BigInteger(num));

return sum.toString().substring(0, 10);

}

public static void main(String[] args) {

System.out.println(sum());

}

}

**14. The following iterative sequence is defined for the set of positive integers:**

n → n/2 (n is even)

n → 3n + 1 (n is odd)

Using the rule above and starting with 13, we generate the following sequence:

13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms.

Although it has not been proved yet (Collatz Problem), it is thought that all starting

numbers finish at 1.

Which starting number, under one million, produces the longest chain?

Ans: 837799

NOTE: Once the chain starts the terms are allowed to go above one million.

import java.util.ArrayList;

public class LongestChain {

public static void main(String[] args) {

ArrayList<Long> list = new ArrayList<Long>();

long length = 0;

int res = 0;

for(int j = 10; j < 1000000; j++)

{

long i = j;

while (i != 1)

{

if (i % 2 == 0)

{

i /= 2;

list.add(i);

}

else

{

i = (3 \* i) + 1;

list.add(i);

}

}

if(list.size() > length)

{

length = list.size();

res = j;

}

list.clear();

}

System.out.println(res);

}

}

15. **Starting in the top left corner of a 2×2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.**

How many such routes are there through a 20×20 grid?

Ans: 137846528820

public class FindingRoutes {

public static long binomialCoefficient(int n, int k)

{

if (k > (n-k)) k = n - k;

long c = 1;

for (int i = 0; i < k; i++)

{

c = c \* (n-i);

c = c / (i+1);

}

return c;

}

public static void main (String[] args)

{

System.out.println(binomialCoefficient(40,20));

}

}

16. **2 power 15 = 32768 and the sum of its digits is 3 + 2 + 7 + 6 + 8 = 26.**

**What is the sum of the digits of the number 2 power 1000? Ans: 1366**

import java.math.BigInteger;

public class SumofDigitsPow {

public static void main(String[] args) {

String temp = BigInteger.ONE.shiftLeft(1000).toString();

int sum = 0;

for(int i=0;i<temp.length();i++){

sum+= temp.charAt(i) - '0';

}

System.out.println(Integer.toString(sum));

}

}

**17. If the numbers 1 to 5 are written out in words: one, two, three, four, five, then there**

**are 3 + 3 + 5 + 4 + 4 = 19 letters used in total.**

If all the numbers from 1 to 1000 (one thousand) inclusive were written out in words,

how many letters would be used?

NOTE: Do not count spaces or hyphens. For example, 342 (three hundred and

forty-two) contains 23 letters and 115 (one hundred and fifteen) contains 20 letters.

The use of "and" when writing out numbers is in compliance with British usage.

Ans: 21124

public class LettersUsed {

private static final String[] ONES = {

"zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine",

"ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"};

private static final String[] TENS = {

"", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"};

private static String toWords(int n) {

if (0 <= n && n < 20)

return ONES[n];

else if (20 <= n && n < 100)

return TENS[n / 10] + (n % 10 != 0 ? ONES[n % 10] : "");

else if (100 <= n && n < 1000)

return ONES[n / 100] + "hundred" + (n % 100 != 0 ? "and" + toWords(n % 100) : "");

else if (1000 <= n && n < 1000000)

return toWords(n / 1000) + "thousand" + (n % 1000 != 0 ? toWords(n % 1000) : "");

else

throw new IllegalArgumentException();

}

public static String NuminWords(int n) {

int sum = 0;

for (int i = 1; i <= n; i++)

sum += toWords(i).length();

return Integer.toString(sum);

}

public static void main(String[] args) {

System.out.println(NuminWords(1000));

}

}

**18. By starting at the top of the triangle below and moving to adjacent numbers on the**

**row below, the maximum total from top to bottom is 23.**

public class MaximumTotalTriangle {

public static void main(String[] args) {

int[][] triangle = { // Mutable

{75},

{95,64},

{17,47,82},

{18,35,87,10},

{20, 4,82,47,65},

{19, 1,23,75, 3,34},

{88, 2,77,73, 7,63,67},

{99,65, 4,28, 6,16,70,92},

{41,41,26,56,83,40,80,70,33},

{41,48,72,33,47,32,37,16,94,29},

{53,71,44,65,25,43,91,52,97,51,14},

{70,11,33,28,77,73,17,78,39,68,17,57},

{91,71,52,38,17,14,91,43,58,50,27,29,48},

{63,66, 4,68,89,53,67,30,73,16,69,87,40,31},

{ 4,62,98,27,23, 9,70,98,73,93,38,53,60, 4,23},

};

for (int i = triangle.length - 2; i >= 0; i--) {

for (int j = 0; j < triangle[i].length; j++)

triangle[i][j] += Math.max(triangle[i + 1][j], triangle[i + 1][j + 1]);

}

System.out.println(Integer.toString(triangle[0][0]));

}

}

**19. You are given the following information, but you may prefer to do some research for yourself**

● 1 Jan 1900 was a Monday.

● Thirty days has September,

● April, June and November.

● All the rest have thirty-one,

● Saving February alone,

● Which has twenty-eight, rain or shine.

● And on leap years, twenty-nine.

● A leap year occurs on any year evenly divisible by 4, but not on a century

unless it is divisible by 400.

How many Sundays fell on the first of the month during the twentieth century (1 Jan

1901 to 31 Dec 2000)?

Ans: 171

public class HowManySundays {

public static int NoofSundays(int ystart, int yend){

int count = 0;

while(ystart <= yend) {

for (int m = 1; m <= 12; m++) {

if(dayOfWeek(ystart, m,1)==0){ //sunday

count++;

}

}

ystart++;

}

return count;

}

public static int dayOfWeek(int year,int month,int day){

int m = (month - 3 + 4800) % 4800;

int y = (year + m / 12) % 400;

m %= 12;

int d = (y + y/4 - y/100 + (13 \* m + 2) / 5 + day + 2) % 7;

return d;

}

public static void main(String[] args) {

System.out.println(NoofSundays(1901, 2000));

}

}

**20. n! means n × (n − 1) × ... × 3 × 2 × 1**

For example, 10! = 10 × 9 × ... × 3 × 2 × 1 = 3628800, and the sum of the digits in the number 10! Is 3 + 6 + 2 + 8 + 8 + 0 + 0 = 27.

Find the sum of the digits in the number 100!

Ans: 648

import java.math.BigInteger;

public class SumofDigits {

public static int SumDigits(String s){

int sum = 0;

for (int i = 0; i < s.length(); i++) {

int temp = Integer.parseInt(s.substring(i, i+1));

sum +=temp;

}

return sum;

}

public static void main(String[] args) {

BigInteger fact = BigInteger.valueOf(1);

for (int i = 1; i <=100; i++) {

fact = fact.multiply(BigInteger.valueOf(i));

}

System.out.println(SumDigits(fact.toString()));

}

}