Binary exponentiation

Binary (binary) exponentiation - is a technique that allows to build any number of in n-th power for $O(\log n)$multiplications (instead of $n$the usual approach of multiplications).

Furthermore, described herein is applicable to any reception **associative** operation and not only to the multiplication number. Recall operation is called associative if for any a, b, cexecuted:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

The most obvious generalization - the remains of some modulo (obviously, the associativity is preserved). Next in "popularity" is a generalization to the matrix product (it is well-known associativity).

Algorithm

Note that for any number $a$and **even** number $n$doable obvious identity (which follows from the associativity of multiplication)

$$a^n = (a^{n/2})^2 = a^{n/2} \cdot a^{n/2}$$

It is the main method in binary exponentiation. Indeed, even $n$we showed how spending just one multiplication operation, you can reduce the problem to a lesser extent by half.

It remains to understand what to do if the power **is odd** . Here we do is very simple: go to the extent of which will have an even:n $n-1$

$$a^n = a^{n-1} \cdot a$$

So, we actually found the recurrence formula: the degree $n$we move, if it is even, to $n/2$and otherwise - to $n-1$. understood that there will be no more $2 \log n$transitions before we arrive $n = 0$(based recursive formula). Thus, we have an algorithm that works for $O(\log n)$multiplications.

Implementation

A simple recursive implementation:

```
int binpow (int a, int n) {
    if (n == 0)
        return 1;
    if (n % 2 == 1)
```

```
          return binpow (a, n-1) * a;
     else {
          int b = binpow (a, n/2);
          return b * b;
     }
}
```

Non-recursive implementation, also optimized (division by 2 replaced bit operations):

```
int binpow (int a, int n) {
     int res = 1;
     while (n)
          if (n & 1) {
               res *= a;
               --n;
          }
          else {
               a *= a;
               n >>= 1;
          }
     return res;
}
```

This implementation can be further simplified somewhat by noting that the construction of a the square is always performed, regardless of the condition worked odd n or not:

```
int binpow (int a, int n) {
     int res = 1;
     while (n) {
          if (n & 1)
               res *= a;
          a *= a;
          n >>= 1;
     }
     return res;
}
```

Finally, it is worth noting that the binary exponentiation is implemented by Java, but only for a class of long arithmetic BigInteger (the pow this class works under binary exponentiation algorithm).

Examples of solving problems

Efficient calculation of Fibonacci numbers

Condition . Given the number n. Required to calculate $F_n$where F_i- the Fibonacci sequence .

Decision . More details are described in the decision paper on the Fibonacci sequence . Here, we briefly present the essence of this decision.

The basic idea is as follows. Calculating the next Fibonacci number is based on the knowledge of the previous two Fibonacci numbers, namely, every next Fibonacci number is the sum of the previous two. This means that we can construct a matrix 2×2that will fit this transformation: as the two Fibonacci numbers $F_i$and F_{i+1}calculate the next number, ie go to the pair F_{i+1}, F_{i+2}. For example, applying this transformation $n$to a couple times $F_0$and F_1we get a couple $F_n$and F_{n+1}. Thus, elevating this transformation matrix n-th power, we thus find the desired $F_n$time for $O(\log n)$what we required.

Erection reshuffle k-th power

Condition . Given permutation $p$length n. Required to build it in k-th power, ie find what happens if to the identity permutation $k$permutation times apply p.

Decision . Simply apply to the permutation $p$algorithm described above binary exponentiation. No differences compared with the erection of the power of numbers - no. Solution is obtained with the asymptotic behavior $O(n \log k)$.

(Note: This problem can be solved more efficiently, **in linear time** . Just select all the cycles in the permutation, and then consider separately each cycle and taking $k$modulo length of the current cycle, to find an answer for this cycle.)

Prompt application of a set of geometric operations to points

Condition . Given $n$points p_i, and given $m$transformation that should be applied to each of these points. Each transformation - is offset by a predetermined or vector or scaling (multiplication coefficients set of coordinates), or around a predetermined rotation axis by a predetermined angle. Furthermore, there is a composite activity cyclic repetition: it has the form "repeat a specified number of times a given list of change" (cyclical repetition of operations can be nested).

Required to calculate the result of the application of these operations to all points (effectively, ie in a time shorter than $O(n \cdot length)$where $length$- the total number of operations that must be done).

Decision . Look at the different types of transformations in terms of how they change the coordinates:

- Shift operation - it just adds to all the coordinates of the unit, multiplication by some constants.
- Zoom operation - it multiplies each coordinate by a constant.
- Rotation operation around the axis - can be represented as follows: new coordinates obtained can be written as a linear combination of the old.
- (We will not specify how this is done. Example, you can submit it for simplicity as a combination of five-dimensional rotations: first, in the planes OXY, and OXZ so that the axis of rotation coincides with the positive direction of the axis OX, then the desired rotation around an axis in the plane YZ, then reverse rotations in planes $OXZ$and OXYso that the rotation axis back to its starting position.)

Easy to see that each of these transformations - this recalculation of coordinates on linear formulas. Thus, any such transformation can be written in matrix form 4×4:

$$\begin{pmatrix} a_1 1 & a_{12} & a_{13} & a_{14} \\ a_2 1 & a_{22} & a_{23} & a_{24} \\ a_3 1 & a_{32} & a_{33} & a_{34} \\ a_4 1 & a_{42} & a_{43} & a_{44} \end{pmatrix},$$

which when multiplied (left) to the line of the old coordinates and constant unit gives a string of new coordinates and constants units:

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} = \begin{pmatrix} x' & y' & z' & 1 \end{pmatrix}.$$

(Why it took to enter a dummy fourth coordinate is always equal to unity? Without this we would have to implement the shift operation: after shift - this is addition to the coordinates of the unit, multiplication by some factors. Without dummy units we would be able to implement only linear combinations of the coordinates themselves, and add to them are given constants - could not.)

Now the solution of the problem becomes almost trivial. Times each elementary operation described by the matrix, the sequence of operations described product of these matrices, and the operation of a cyclic repetition - the erection of this matrix to a power. Thus, during the time we $O(m \cdot \log repetition)$can predposchitat matrix 4×4describing all the changes and then simply multiply each point $p_i$on the matrix - thus, we will answer all requests for time $O(n)$.

The number of paths in a fixed-length box

Condition . Given an undirected graph $G$with $n$vertices, and given a number k . Required for each pair of vertices $i$and $j$find the number of paths between them containing exactly $k$edges.

Decision . More details on this problem is considered in a separate article . Here we recall only the essence of this solution: we simply erect a k-th power of the adjacency matrix of the graph, and the matrix elements and the decisions will be. Total asymptotics - $O(n^3 \log k)$.

(Note: In the same article, and considered another variation of this problem: when the weighted graph, and want to find the path of minimum weight, containing exactly $k$the edges. As shown in this paper, this problem is also solved by using binary exponentiation adjacency matrix, but instead of the normal operation of multiplying two matrices should use a modified: instead of multiplying the amount taken, and instead of summing - taking a minimum.)

Variation binary exponentiation: multiplication of two numbers modulo

We present here an interesting variation of the binary exponentiation.

Suppose we are faced with such a **task** : to multiply two numbers $a$and $b$modulo m:

$$a \cdot b \pmod m$$

Assume that the numbers can be quite large: so that the numbers themselves are placed in a built-in data types, but their direct product $a \cdot b$- no longer exists (note that we also need to sum numbers placed in a built-in data type). Accordingly, the task is to calculate the required value $(a \cdot b) \pmod m$, without the aid ofa long arithmetic .

Solution is as follows. We simply apply the binary exponentiation algorithm described above, but instead of multiplication, we will make the addition. In other words, the multiplication of two numbers, we have reduced to $O(\log m)$the operations of addition and multiplication by two (which is also, in fact, there is addition).

(Note: This problem can be solved **in another way** , by resorting to assistance operations with floating-point numbers. Namely, to calculate a float expression $a \cdot b / m$, and round it to the nearest integer. So we find **an approximate** quotient. rescued him from work $a \cdot b$(ignoring overflow), we are likely to get a relatively small number, which can be taken modulo m- and return it as a response. This solution looks quite fragile, but it is very fast, and very briefly implemented.)

Problem in online judges

List of tasks that can be solved using binary exponentiation:

- SGU # 265 **"Wizards"** [Difficulty: Medium]

url - > http://e-maxx.ru/algo/binary_pow