Ошибка 1

1. Зубарева Наталия Дмитриевна;
2. 17/09/2021 22:08;
3. При любых входных параметрах функция возвращает 0, так как, в отличие от описания реализации (*r*-1 = 1), изначальное значение r=0;
4. Нарушается требование 2 и имеется несоответствие описанию реализации;
5. При любых значениях первого и второго аргументов метод возвращает 0, в то время как, согласно требованию, должен возвращать результат возведения первого аргумента в степень, равную второму, с некоторыми уточнениями. Кроме положительных степеней нуля и степеней равных 2^32 никакая другая степень не должна быть равна нулю, так что это ошибка. Кроме того, согласно описанию алгоритма, изначальное значение r должно быть равно 1, но оно равно 0, что является несоблюдением требования к алгоритму;
6. for(int i = -10; i < 11; ++i){

for(int j = -10; j < 11; ++j){

System.out.println(pow(i, j));

}

} // выводит все 0, при том, что так быть не должно

1. Чтобы исправить ошибку, в самом начале метода нужно исправить выражение int r = 0; на int r = 1; в соответствии с начальным шагом алгоритма. При нулевом r дальнейшее умножение на a бесполезно, так как, возвращая r, программа всегда будет возвращать 0. Если r будет начинаться с 1, то далее его значение все-таки будет изменяться.

Ошибка 2

1. Зубарева Наталия Дмитриевна;
2. 17/09/2021 22:20;
3. При некоторых положительных значениях второго аргумента получается неверный результат;
4. Нарушается требование 2;
5. Происходят ошибки при вычислении положительных степеней, кроме четных степеней, двоичное представление которых оканчивается на ровно один 0 и симметрично, если его отбросить, так как в цикле программа умножает r на r. При нечетных степенях значение очень быстро растет, вместо первой степени числа при вычислении первой степени вычисляется квадрат, далее ошибка увеличивается, на 17 степени происходит переполнение. При четных степенях, как правило, получается результат меньше требуемой степени, но не всегда. При четных степенях, которые в бинарном виде симметричны после отбрасывания одного правого нуля, результат получается верным до тех пор, пока не наступит переполнение (то есть до 30 включительно);
6. for(int i = 0; i < 20; ++i){

System.out.println(pow(2, i));

}// результаты иногда совпадают с требуемыми степенями двойки, но чаще нет

1. Ошибка происходит из-за того, что вместо умножения a на себя написано умножение r на себя. Чтобы ее исправить, в цикле необходимо исправить r \*= r; на умножение a \*= a, в соответствии с описанием работы алгоритма;

Ошибка 3

1. Зубарева Наталия Дмитриевна;
2. 17/09/2021 20:56;
3. Некорректные требования к сигнатуре и переполнениям;
4. Нарушается требование 2с;
5. Согласно требованиям, переполнения должны обрабатываться так: если результат превосходит по модулю 2^31, стоит возвращать инт, полученный как результат, взятый по модулю 2^32. Вероятно, это сделано, чтобы не получать странных отрицательных результатов там, где не надо. Однако, эти требования не учитывают ситуации переполнения инт в языке java: переполнение происходит уже на числе 2^31, но все числа от него до 2^32 не имеет смысла брать по модулю 2^32 для исправления ситуации. Так, при входных параметрах 2 и 31 функция вычислит результат -2147483648, хотя должна, казалось бы, возвращать что-то другое;
6. System.out.println(pow(2,31));//получаем отрицательное число, не очень здорово;
7. Для обработки переполнения в соответствии с требованием 2с можно поменять сигнатуру функции на то, чтобы она возвращала лонг, принимала а как лонг, сделать r типа лонг, и возвращать результат по модулю 2^32.

public long pow(long a, int b) {

long r = 1;

while(b > 0) {

if((b&1) != 0) r \*= a;

a \*= a;

b >>= 1;

}

return r&0xffffffff;

}

Альтернативно, и, как мне кажется, лучше было бы изменить требование 2с и брать ответ по модулю 2^31, чтобы сделать обработку переполнений в соответствии с моментом переполнения в языке. Тогда остается интовая сигнатура, а вычисляется по модулю 2^31 на каждой итерации и ответ возвращается по модулю 2^31:

public int pow(int a, int b) {

int r = 1;

while(b > 0) {

if((b&1) != 0) r \*= a;

a&=0x7fffffff;

a \*= a;

b >>= 1;

}

return r&0x7fffffff;

}