Find the Closest Palindrome

Given an integer n, find the closest integer (not including itself), which is a palindrome.

The 'closest' is defined as absolute difference minimized between two integers.

Example 1:

```
Input: "123"
Output: "121"
```

Note:

1. The input **n** is a positive integer represented by string, whose length will not exceed 18.
2. If there is a tie, return the smaller one as answer.

Solution 1

Let's build a list of candidate answers for which the final answer must be one of those candidates. Afterwards, choosing from these candidates is straightforward.

If the final answer has the same number of digits as the input string `S`, then the answer must be the middle digits + (-1, 0, or 1) flipped into a palindrome. For example, `23456` had middle part `234`, and `233, 234, 235` flipped into a palindrome yields `23332, 23432, 23532`. Given that we know the number of digits, the prefix `235` (for example) uniquely determines the corresponding palindrome `23532`, so all palindromes with larger prefix like `23732` are strictly farther away from S than `23532 >= S`.

If the final answer has a different number of digits, it must be of the form `999....999` or `1000...0001`, as any palindrome smaller than `99....99` or bigger than `100....001` will be farther away from S.

```
def nearestPalindromic(self, S):
    K = len(S)
    candidates = [str(10**k + d) for k in (K-1, K) for d in (-1, 1)]
    prefix = S[:(K+1)/2]
    P = int(prefix)
    for start in map(str, (P-1, P, P+1)):
        candidates.append(start + (start[:-1] if K%2 else start)[::-1])

    def delta(x):
        return abs(int(S) - int(x))

    ans = None
    for cand in candidates:
        if cand != S and not cand.startswith('00'):
            if (ans is None or delta(cand) < delta(ans) or
                    delta(cand) == delta(ans) and int(cand) < int(ans)):
                ans = cand
    return ans
```

written by awice original link here

Solution 2

```
public class Solution {
    public String nearestPalindromic(String n) {
        if (n.length() >= 2 && allNine(n)) {
            String s = "1";
            for (int i = 0; i < n.length() - 1; i++) {
                s += "0";
            }
            s += "1";
            return s;
        }
        boolean isOdd = (n.length() % 2 != 0);
        String left = n.substring(0, (n.length() + 1) / 2);
        long[] increment = {-1, 0, +1};
        String ret = n;
        long minDiff = Long.MAX_VALUE;
        for (long i : increment) {
            String s = getPalindrom(Long.toString(Long.parseLong(left) + i), isOdd)
;
            if (n.length() >= 2 && (s.length() != n.length() || Long.parseLong(s) ==
0)) {
                s = "";
                for (int j = 0; j < n.length() - 1; j++) {
                    s += "9";
                }
            }
            long diff = s.equals(n) ? Long.MAX_VALUE : Math.abs(Long.parseLong(s) -
Long.parseLong(n));
            if (diff < minDiff) {
                minDiff = diff;
                ret = s;
            }
        }
        return ret;
    }
    private String getPalindrom(String s, boolean isOdd) {
        String right = new StringBuilder(s).reverse().toString();
        return isOdd ? s.substring(0, s.length() - 1) + right : s + right;
    }
    private boolean allNine(String s) {
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) != '9') {
                return false;
            }
        }
        return true;
    }
}
```

written by 2499370956 original link here

Solution 3

My solution take @uwi 's solution as reference, and add my understanding to it.
We first need to find the higher palindrome and lower palidrome respectively. and return the one who has the least different with the input number.

For the higher palindrome, the low limit is number + 1 while for the lower palindrome, the high limit is number - 1.

One global solution to find a palindrome is to copy first half part of the array to the last half part, we regards this as standard palindrome.
We need to detect this standard palindrome belongs to higher one or the lower one. And other solutions will be based on this standard one.

For the higher palindrome, if the standard one belongs to higher, we just simply return it. Or we need to change it.
For example, String n is 1343, and the standard palindrome is 1331. to get the higher one from the standard palidrome, we start from the
first 3, which is (n.length - 1) / 2. Add the number by 1, (---> 1431)if the added result is not higher than 9, the changing process is finished,
otherwise, continuously changing the number of previous index by i. After the changing process, we re-palidrome the string. (1431 -- > 1441)

For the lower palindrome, similar idea.But we need to notice that when we decrease a number, and if the first character of the string is '0',
we need to resize the array of n.length - 1, and fill in with '9'. (for example, n is '1000', the standard palidrome is '1001'(higher one) ,the lower one '0000'-->'999'.)

```
public class Solution {
    public String nearestPalindromic(String n) {
        Long number = Long.parseLong(n);
        Long big = findHigherPalindrome(number + 1);
        Long small = findLowerPalindrome(number - 1);
        return Math.abs(number - small) > Math.abs(big - number) ? String.valueOf(b
ig) : String.valueOf(small);
    }
    public Long findHigherPalindrome(Long limit) {
        String n = Long.toString(limit);
        char[] s = n.toCharArray(); // limit
        int m = s.length;
        char[] t = Arrays.copyOf(s, m); // target
        for (int i = 0; i < m / 2; i++) {
            t[m - 1 - i] = t[i];
        }
        for (int i = 0; i < m; i++) {
            if (s[i] < t[i]) {
                return Long.parseLong(String.valueOf(t));
            } else if (s[i] > t[i]) {
                for (int j = (m - 1) / 2; j >= 0; j--) {
                    if (++t[j] > '9') {
                        t[j] = '0';
                    } else {
                        break;
                    }
```

```
                }
                // make it palindrome again
                for (int k = 0; k < m / 2; k++) {
                    t[m - 1 - k] = t[k];
                }
                return Long.parseLong(String.valueOf(t));
            }
        }
        return Long.parseLong(String.valueOf(t));
    }
    public Long findLowerPalindrome(Long limit) {
        String n = Long.toString(limit);
        char[] s = n.toCharArray();
        int m = s.length;
        char[] t = Arrays.copyOf(s, m);
        for (int i = 0; i < m / 2; i++) {
            t[m - 1 - i] = t[i];
        }
        for (int i = 0; i < m; i++) {
            if (s[i] > t[i]) {
                return Long.parseLong(String.valueOf(t));
            } else if (s[i] < t[i]) {
                for (int j = (m - 1) / 2; j >= 0; j--) {
                    if (--t[j] < '0') {
                        t[j] = '9';
                    } else {
                        break;
                    }
                }
                if (t[0] == '0') {
                    char[] a = new char[m - 1];
                    Arrays.fill(a, '9');
                    return Long.parseLong(String.valueOf(a));
                }
                // make it palindrome again
                for (int k = 0; k < m / 2; k++) {
                    t[m - 1 - k] = t[k];
                }
                return Long.parseLong(String.valueOf(t));
            }
        }
         return Long.parseLong(String.valueOf(t));
    }
}
```

written by yuhengw original link here