Manacher算法  
【题目】  
给定一个字符串str，返回str中的最长回文子串的长度。  
【举例】  
str=“123”。其中的最长回文子串“1”或者“2”或者“3”，所以返回1。  
str=“abc1234321ab”。其中的最长回文子串“1234321”，所以返回7。  
【进阶题目】  
给定一个字符串str，想通过添加字符的方式使得str整体都变成回文字符串，但要求只能在str的末尾添加字符，请返回在str后面添加的最短字符串。  
【举例】  
str=“12”。在末尾添加“1”之后，str变为“121”是回文串。在末尾添加“21”之后，str变为“1221”也是回文串。但“1”是所有添加方案中最短的，所以返回“1”。  
【要求】  
如果str长度为N，解决原问题和进阶问题的时间复杂度都达到O(N)。  
  
  
  
原问题代码：  
  
    public char[] manacherString(String str) {  
        char[] charArr = str.toCharArray();  
        char[] res = new char[str.length() \* 2 + 1];  
        int index = 0;  
        for (int i = 0; i != res.length; i++) {  
            res[i] = (i & 1) == 0 ? '#' : charArr[index++];  
        }  
        return res;  
    }  
  
    public int maxLcpsLength(String str) {  
        if (str == null || str.length() == 0) {  
            return 0;  
        }  
        char[] charArr = manacherString(str);  
        int[] pArr = new int[charArr.length];  
        int index = -1;  
        int pR = -1;  
        int max = Integer.MIN\_VALUE;  
        for (int i = 0; i != charArr.length; i++) {  
            pArr[i] = pR > i ? Math.min(pArr[2 \* index - i], pR - i) : 1;  
            while (i + pArr[i] < charArr.length && i - pArr[i] > -1) {  
                if (charArr[i + pArr[i]] == charArr[i - pArr[i]])  
                    pArr[i]++;  
                else {  
                    break;  
                }  
            }  
            if (i + pArr[i] > pR) {  
                pR = i + pArr[i];  
                index = i;  
            }  
            max = Math.max(max, pArr[i]);  
        }  
        return max - 1;  
    }  
  
进阶问题代码：  
  
    public String shortestEnd(String str) {  
        if (str == null || str.length() == 0) {  
            return null;  
        }  
        char[] charArr = manacherString(str);  
        int[] pArr = new int[charArr.length];  
        int index = -1;  
        int pR = -1;  
        int maxContainsEnd = -1;  
        for (int i = 0; i != charArr.length; i++) {  
            pArr[i] = pR > i ? Math.min(pArr[2 \* index - i], pR - i) : 1;  
            while (i + pArr[i] < charArr.length && i - pArr[i] > -1) {  
                if (charArr[i + pArr[i]] == charArr[i - pArr[i]])  
                    pArr[i]++;  
                else {  
                    break;  
                }  
            }  
            if (i + pArr[i] > pR) {  
                pR = i + pArr[i];  
                index = i;  
            }  
            if (pR == charArr.length) {  
                maxContainsEnd = pArr[i];  
                break;  
            }  
        }  
        char[] res = new char[str.length() - maxContainsEnd + 1];  
        for (int i = 0; i < res.length; i++) {  
            res[res.length - 1 - i] = charArr[i \* 2 + 1];  
        }  
        return String.valueOf(res);  
    }

bfprt算法及其相关  
找到无序数组中最小的K个数  
【题目】  
给定一个无序的整型数组arr，找到其中最小的k个数。  
【要求】  
如果数组arr的长度为N，排序之后自然可以得到最小的k个数，此时时间复杂度为排序的时间复杂度即O(N\*logN)。本题要求读者实现时间复杂度O(N\*logK)和O(N)的方法。  
  
  
利用堆：  
  
    public int[] getMinKNumsByHeap(int[] arr, int k) {  
        if (k < 1 || k > arr.length) {  
            return arr;  
        }  
        int[] kHeap = new int[k];  
        for (int i = 0; i != k; i++) {  
            heapInsert(kHeap, arr[i], i);  
        }  
        for (int i = k; i != arr.length; i++) {  
            if (arr[i] < kHeap[0]) {  
                kHeap[0] = arr[i];  
                heapify(kHeap, 0, k);  
            }  
        }  
        return kHeap;  
    }  
  
    public void heapInsert(int[] arr, int value, int index) {  
        arr[index] = value;  
        while (index != 0) {  
            int parent = (index - 1) / 2;  
            if (arr[parent] < arr[index]) {  
                swap(arr, parent, index);  
                index = parent;  
            } else {  
                break;  
            }  
        }  
    }  
  
    public void heapify(int[] arr, int index, int heapSize) {  
        int left = index \* 2 + 1;  
        int right = index \* 2 + 2;  
        int largest = index;  
        while (left < heapSize) {  
            if (arr[left] > arr[index]) {  
                largest = left;  
            }  
            if (right < heapSize && arr[right] > arr[largest]) {  
                largest = right;  
            }  
            if (largest != index) {  
                swap(arr, largest, index);  
            } else {  
                break;  
            }  
            index = largest;  
            left = index \* 2 + 1;  
            right = index \* 2 + 2;  
        }  
    }  
  
    public void swap(int[] arr, int index1, int index2) {  
        int tmp = arr[index1];  
        arr[index1] = arr[index2];  
        arr[index2] = tmp;  
    }  
  
  
利用bfprt算法：  
  
  
  
    public int[] getMinKNumsByBFPRT(int[] arr, int k) {  
        if (k < 1 || k > arr.length) {  
            return arr;  
        }  
        int minKth = getMinKthByBFPRT(arr, k);  
        int[] res = new int[k];  
        int index = 0;  
        for (int i = 0; i != arr.length; i++) {  
            if (arr[i] < minKth) {  
                res[index++] = arr[i];  
            }  
        }  
        for (; index != res.length; index++) {  
            res[index] = minKth;  
        }  
        return res;  
    }  
  
    public int getMinKthByBFPRT(int[] arr, int K) {  
        int[] copyArr = copyArray(arr);  
        return select(copyArr, 0, copyArr.length - 1, K - 1);  
    }  
  
    public int[] copyArray(int[] arr) {  
        int[] res = new int[arr.length];  
        for (int i = 0; i != res.length; i++) {  
            res[i] = arr[i];  
        }  
        return res;  
    }  
  
    public int select(int[] arr, int begin, int end, int i) {  
        if (begin == end) {  
            return arr[begin];  
        }  
        int pivot = medianOfMedians(arr, begin, end);  
        int[] pivotRange = partition(arr, begin, end, pivot);  
        if (i >= pivotRange[0] && i <= pivotRange[1]) {  
            return arr[i];  
        } else if (i < pivotRange[0]) {  
            return select(arr, begin, pivotRange[0] - 1, i);  
        } else {  
            return select(arr, pivotRange[1] + 1, end, i);  
        }  
    }  
  
    public int medianOfMedians(int[] arr, int begin, int end) {  
        int num = end - begin + 1;  
        int offset = num % 5 == 0 ? 0 : 1;  
        int[] mArr = new int[num / 5 + offset];  
        for (int i = 0; i < mArr.length; i++) {  
            int beginI = begin + i \* 5;  
            int endI = beginI + 4;  
            mArr[i] = getMedian(arr, beginI, Math.min(end, endI));  
        }  
        return select(mArr, 0, mArr.length - 1, mArr.length / 2);  
    }  
  
    public int[] partition(int[] arr, int begin, int end, int pivotValue) {  
        int small = begin - 1;  
        int cur = begin;  
        int big = end + 1;  
        while (cur != big) {  
            if (arr[cur] < pivotValue) {  
                swap(arr, ++small, cur++);  
            } else if (arr[cur] > pivotValue) {  
                swap(arr, cur, --big);  
            } else {  
                cur++;  
            }  
        }  
        int[] range = new int[2];  
        range[0] = small + 1;  
        range[1] = big - 1;  
        return range;  
    }  
  
    public int getMedian(int[] arr, int begin, int end) {  
        insertionSort(arr, begin, end);  
        int sum = end + begin;  
        int mid = (sum / 2) + (sum % 2);  
        return arr[mid];  
    }  
  
    public void insertionSort(int[] arr, int begin, int end) {  
        for (int i = begin + 1; i != end + 1; i++) {  
            for (int j = i; j != begin; j--) {  
                if (arr[j - 1] > arr[j]) {  
                    swap(arr, j - 1, j);  
                } else {  
                    break;  
                }  
            }  
        }  
    }  
  
    public void swap(int[] arr, int index1, int index2) {  
        int tmp = arr[index1];  
        arr[index1] = arr[index2];  
        arr[index2] = tmp;  
    }   
KMP算法  
【题目】  
给定两个字符串str和match，长度分别为N和M。实现一个算法，如果字符串str中含有字串match，则返回match在str中的开始位置，不含有则返回-1。  
【举例】  
str=“acbc”，match=“bc”。返回2。  
str=“acbc”，match=“bcc”。返回-1。  
【要求】  
如果match的长度大于str长度(M>N)，str必然不会含有match，可直接返回-1。但如果N>=M，要求算法复杂度O(N)。  
  
    public int getIndexOf(String s, String m) {  
        if (s == null || m == null || m.length() < 1 || s.length() < m.length()) {  
        return -1;  
        }  
        char[] ss = s.toCharArray();  
        char[] ms = m.toCharArray();  
        int si = 0;  
        int mi = 0;  
        int[] next = getNextArray(ms);  
        while (si < ss.length && mi < ms.length) {  
        if (ss[si] == ms[mi]) {  
            si++;  
            mi++;  
        } else if (next[mi] == -1) {  
            si++;  
        } else {  
            mi = next[mi];  
        }  
        }  
        return mi == ms.length ? si - mi : -1;  
    }  
  
  
    public int[] getNextArray(char[] ms) {  
        if (ms.length == 1) {  
            return new int[] { -1 };  
        }  
        int[] next = new int[ms.length];  
        next[0] = -1;  
        next[1] = 0;  
        int pos = 2;  
        int cn = 0;  
        while (pos < next.length) {  
            if (ms[pos - 1] == ms[cn]) {  
                next[pos++] = ++cn;  
            } else if (cn > 0) {  
                cn = next[cn];  
            } else {  
                next[pos++] = 0;  
            }  
        }  
        return next;  
    }