#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

/\*

 \* Complete the 'highestValuePalindrome' function below.

 \*

 \* The function is expected to return a STRING.

 \* The function accepts following parameters:

 \*  1. STRING s

 \*  2. INTEGER n

 \*  3. INTEGER k

 \*/

/\*

 \* To return the string from the function, you should either do static allocation or dynamic allocation

 \*

 \* For example,

 \* char\* return\_string\_using\_static\_allocation() {

 \*     static char s[] = "static allocation of string";

 \*

 \*     return s;

 \* }

 \*

 \* char\* return\_string\_using\_dynamic\_allocation() {

 \*     char\* s = malloc(100 \* sizeof(char));

 \*

 \*     s = "dynamic allocation of string";

 \*

 \*     return s;

 \* }

 \*

 \*/

char\* highestValuePalindrome(char\* s, int n, int k) {

char \*result=strdup(s);

int \*changed=calloc(n,sizeof(int));

int used=0;

for(int i=0;i<n/2;i++){

    int j=n-1-i;

    if(result[i]!=result[j]){

        if(result[i]>result[j]){

            result[j]=result[i];

        }else{

            result[i]=result[j];

        }

        changed[i]=changed[j]=1;

        used++;

    }

}

if(used>k){

    free(changed);

    free(result);

    return strdup("-1");

}

int remaining=k-used;

for(int i=0;i<n/2 && remaining>0;i++){

    int j=n-1-i;

    if(result[i]!='9'){

        if(changed[i]||changed[j]){

            if(remaining>=1){

                result[i]=result[j]='9';

                remaining-=1;

            }

        }else{

            if(remaining >=2){

result[i]=result[j]='9';

remaining-=2;

            }

        }

    }

}

if(n%2==1 && remaining >0){

    result[n/2]='9';

}

free(changed);

return result;

}

int main()

{

    FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

    char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

    int n = parse\_int(\*(first\_multiple\_input + 0));

    int k = parse\_int(\*(first\_multiple\_input + 1));

    char\* s = readline();

    char\* result = highestValuePalindrome(s, n, k);

    fprintf(fptr, "%s\n", result);

    fclose(fptr);

    return 0;

}

char\* readline() {

    size\_t alloc\_length = 1024;

    size\_t data\_length = 0;

    char\* data = malloc(alloc\_length);

    while (true) {

        char\* cursor = data + data\_length;

        char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

        if (!line) {

            break;

        }

        data\_length += strlen(cursor);

        if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

            break;

        }

        alloc\_length <<= 1;

        data = realloc(data, alloc\_length);

        if (!data) {

            data = '\0';

            break;

        }

    }

    if (data[data\_length - 1] == '\n') {

        data[data\_length - 1] = '\0';

        data = realloc(data, data\_length);

        if (!data) {

            data = '\0';

        }

    } else {

        data = realloc(data, data\_length + 1);

        if (!data) {

            data = '\0';

        } else {

            data[data\_length] = '\0';

        }

    }

    return data;

}

char\* ltrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    while (\*str != '\0' && isspace(\*str)) {

        str++;

    }

    return str;

}

char\* rtrim(char\* str) {

    if (!str) {

        return '\0';

    }

    if (!\*str) {

        return str;

    }

    char\* end = str + strlen(str) - 1;

    while (end >= str && isspace(\*end)) {

        end--;

    }

    \*(end + 1) = '\0';

    return str;

}

char\*\* split\_string(char\* str) {

    char\*\* splits = NULL;

    char\* token = strtok(str, " ");

    int spaces = 0;

    while (token) {

        splits = realloc(splits, sizeof(char\*) \* ++spaces);

        if (!splits) {

            return splits;

        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");

    }

    return splits;

}

int parse\_int(char\* str) {

    char\* endptr;

    int value = strtol(str, &endptr, 10);

    if (endptr == str || \*endptr != '\0') {

        exit(EXIT\_FAILURE);

    }

    return value;

}