Highest Value Palindrome

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * 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
 */

string highestValuePalindrome(string s, int n, int k) {
    string res=s;
    vector<int> diff(n,0);
    int l=0,r=n-1;
    while(l<r){
        if(s[l]!=s[r]){
            res[l]=res[r]=max(s[l],s[r]);
            diff[l]=diff[r]=1;
            k--;
        }
        l++;r--;
    }
    if(k<0) return "-1";
    l=0;r=n-1;
    while(l<=r&&k>0){
        if(l==r){
            if(k>0&&res[l]!='9'){res[l]='9';k--;}
        }else{
            if(res[l]!='9'){
                if(diff[l]&&k>=1){res[l]=res[r]='9';k--;}
                else if(!diff[l]&&k>=2){res[l]=res[r]='9';k-=2;}
            }
        }
        l++;r--;
    }
```

```
    return res;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);

    int k = stoi(first_multiple_input[1]);

    string s;
    getline(cin, s);

    string result = highestValuePalindrome(s, n, k);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
```

```
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```