Christmas Party

Let's assume that|X|≤|Y|. Then XY=XPX^r where P = P^r. Let's iterate through possible positions for AX|BPX^rC split, if we fix such position we may calculate for all possible BP|X^rC splits how long might X^r be. To do this we should calculate longest common prefix of X^rA^r and X^rC. This may be done by quadratic dp (assuming that lcp[0][j] = lcp[i][n+1]=0):

```
for(int i = 1; i <= n; i++) {

    for(int j = i + 1; j <= n; j++) {

        lcp[i][j] = (s[i] == s[j]) * (1 + lcp[i - 1][j + 1]);

    }

}
```

Now if we fixed positions i and j of AX|_iBP|_jX^rC splits, we should take the length of longest possible X^r in that position and multiply it with the number of palindromes which start after i and end in j. This also may be done by quadratic dynamic programming. Note that if |X| > |Y| situation is same but we take XY = Y^rPY, thus we should also consider number of prefix-palindromes of string between i and j. It's all calculated as follows:

```
for(int i = 1; i <= n; i++) {

    is_pal[i][i] = 1;

    pre[i][i] = suf[i][i] = 2;

    is_pal[i][i - 1] = 1;

    pre[i][i - 1] = suf[i][i - 1] = 1;

}

for(int len = 1; len < n; len++) {

    for(int i = 1; i + len <= n; i++) {

    is_pal[i][i + len] = (s[i] == s[i + len]) * is_pal[i + 1][i + len - 1];

    pre[i][i + len] = is_pal[i][i + len] + pre[i][i + len - 1];

    suf[i][i + len] = is_pal[i][i + len] + suf[i + 1][i + len];

    }

}
```

Now that we're done with auxiliary dp's, we may calculate the final answer. To do this we consider all possible splits AX|_iBP|_jX^rC and add lcp[i][j]·(pre[i+1][j−1]+suf[i+1][j−1]−1) to the answer. We subtracted 1 here because empty string was counted in both pre[i+1][j-1] and suf[i+1][j-1] but it produces same pairs of strings. So, final solution looks like this:

```
int ans = 0;

for(int i = 1; i <= n; i++) {
```

```
    for(int j = i + 1; j <= n; j++) {
        ans += lcp[i][j] * (pre[i + 1][j - 1] + suf[i + 1][j - 1] - 1);
    }
}
```

Complete Code:-

```
#include <bits/stdc++.h>
using namespace std;
#define all(x) begin(x), end(x)
#define int int64_t
const int maxn = 1001;
int pre[maxn][maxn], suf[maxn][maxn], is_pal[maxn][maxn], lcp[maxn][maxn];
signed main() {
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0);
    string s;
    cin >> s;
    int n = s.size();
    for(int i = 0; i < n; i++) {
        is_pal[i][i] = 1;
        pre[i][i] = suf[i][i] = 2;
        if(i > 0) {
        is_pal[i][i - 1] = pre[i][i - 1] = suf[i][i - 1] = 1;
    }
}
for(int i = 0; i < n; i++) {
    for(int j = i + 1; j < n; j++) {
    lcp[i][j] = (s[i] == s[j]) * (1 + (i > 0 && j + 1 < n ? lcp[i - 1][j + 1] : 0));
    }
```

```
}
for(int len = 1; len < n; len++) {
    for(int i = 0; i + len < n; i++) {
        is_pal[i][i + len] = (s[i] == s[i + len]) * is_pal[i + 1][i + len - 1];
        pre[i][i + len] = is_pal[i][i + len] + pre[i][i + len - 1];
        suf[i][i + len] = is_pal[i][i + len] + suf[i + 1][i + len];
    }
}
int ans = 0;
for(int i = 0; i < n; i++) {
    for(int j = i + 1; j < n; j++) {
        ans += lcp[i][j] * (pre[i + 1][j - 1] + suf[i + 1][j - 1] - 1);
    }
}
    cout << ans << endl;
    return 0;
}
```