```cpp
#include<iostream>
#include<vector>
#include<map>
#include<string>
#include<unordered_map>
#include<string.h>
#include<stdlib.h>

using namespace std;

bool s1[5000][5001] = {{false,false}};

bool isP(string& str) {
    if(str.length() == 1) {
        return true;
    } else {
        return str[0] == str[str.length()-1];
    }
}

void init(string& in) {
    int len = in.length();
    int lmax = len < 3 ? len : 3;

    for(int i = 0; i < len; i++) {
        s1[i][0]=true;
    }
    for(int l = 1; l <= lmax; l++) {
        for(int pos = 0; pos <= (int)len - l; pos++) {
            string subStr = in.substr(pos, l);
            if(isP(subStr)) {
                s1[pos][l]=true;
            }
        }
    }
}

void dp(string& in) {
    int len = in.length();

    for(int l = 4; l <= len; l++) {
        for(int pos=0; pos<=len-l; pos++) {
            s1[pos][l] = (s1[pos+2][l-4] && (in[pos] == in[pos+l-1]));
        }
    }
}

typedef struct _ab {_ab* p; _ab* na; _ab* nb; int count; } AB;
AB* r;
void buildRTree(string& in) {
    r = new AB({0,0,0,0});
    int newA = 0;
    int newB = 0;
    for(int pos = 0; pos < (int) in.length(); pos++) {
        AB* c = r;
        for(int l = 1; l <= (int) in.length() - pos; l++) {
            if(in[pos+l-1] == 'a') {
                if(!c->na) { c->na = new AB({c,0,0,0}); newA++; }
                c = c->na;
                if(s1[pos][l])
                    c->count++;
            } else {
                if(!c->nb) { c->nb = new AB({c,0,0,0}); newB++;}
                c = c->nb;
                if(s1[pos][l])
                    c->count++;
            }
        }
    }
    //cout << newA << "," << newB << endl;
}
string sol;
void traverse(AB* c, int& k) {
    //cout << sol << ',' << c->count << endl;
    if(k<=0) {
        return;
    }
    k -= c->count;
    if(k <= 0) {
        cout << sol << endl;
    }
    if(c->na) {
        sol.push_back('a');
        traverse(c->na, k);
        sol.pop_back();
    }
```

```
    if(c->nb) {
        sol.push_back('b');
        traverse(c->nb, k);
        sol.pop_back();
    }
}

int main() {
    string in;
    int k;
    getline(cin, in);
    cin >> k;

    init(in);
    dp(in);
    buildRTree(in);

    AB* c = r;
    traverse(c, k);
    return 0;
}
```