qiyue的回文树

qiyue平时训练就很喜欢写串串的题目，有一天他学到了一种新的数据结构叫回文树，能一次性找出一个串里的所有回文子串，他觉得很开心，就跑去找kangdi聊天去了，kangdi：“我觉得你这个数据结构不错，不过都2019年了怎么还有人看回文串这种无趣的东西呢……这样子吧，我希望你能扩展一下这个数据结构，找到一个串中所有不完全回文子串。”qiyue:”那是啥…”kangdi:“一个不完全回文子串指的是那个串中奇数位置上字符中心对称所对应的字符相等，比如"abaa", "a", "bb", "abbbaa",“aaa”就是不完全回文子串，"ab", "bba" and "aaabaa"，就啥也不是，我感觉这样的子串挺多的，你别一一列出来了，我只想问你字典序第k小的串是啥,如果一个子串出现了多次就统计多次”qiyue感觉这个问题他没啥头绪，你能帮帮他吗。

字典序：对于两个字符串，显然的做法是先按照第一个字母、以 a、b、c……z 的顺序排列；如果第一个字母一样，那么比较第二个、第三个乃至后面的字母。如果比到最后两个单词不一样长（比如，sigh 和 sight），那么把短者排在前。

输入：

第一行输入一个字符串 字符串长度不超过4000，输入仅有a,b两种字符，由于qiyue不太会造数据，所以数据随机生成。

第二行输入一个整数k，代表询问字典序第k小子串(保证字符串中不完全回文子串不超过k个)

输出：

输出仅一行，字典序第k小子串。

#include <bits/stdc++.h>  
using namespace std;  
bool matrix[5200][5200];  
string s;  
int trie[25100000][2], deep[25100000], fa[25100000], ans[25100000];  
int last[10000];  
char ch[6000];  
int N;  
int len = 0;  
void dfs1(int a, int k)  
{  
if (k == 0)  
{  
matrix[a][a] = true;  
while (a - k > 0 && ch[a - k] == ch[a + k])  
matrix[a - k][a + k] = true, k++;  
if (k != 0)  
k--;  
dfs1(a, k + 2);  
return;  
}  
if (a - k < 0)  
return;  
if (ch[a - k] == ch[a + k])  
{  
matrix[a - k][a + k] = true;  
dfs1(a, k + 2);  
}  
}  
void dfs2(int a, int k)  
{  
if (k == 0)  
{  
while (a - k > 0 && ch[a - k] == ch[a + k-1])  
matrix[a - k][a + k-1] = true, k++;  
if (k != 0)  
k--;  
dfs2(a, k + 2);  
return;  
}  
if (a - k < 0)  
return;  
if (ch[a - k] == ch[a + k-1])  
{  
matrix[a - k][a + k-1] = true;  
dfs2(a, k + 2);  
}  
}  
list<int> st;  
int K;  
bool dfs3(int p)  
{  
//cout << K <<" "<<ans[p]<< endl;  
K -= ans[p];  
if (K < 1)  
return true;  
for (int i = 0; i < 2; ++i)  
{  
if (trie[p][i] != -1)  
{  
st.push\_back(i);  
if (dfs3(trie[p][i]))  
return true;  
st.pop\_back();  
}  
}  
return false;  
}  
int main()  
{  
ios::sync\_with\_stdio(false);  
cin.tie(NULL);  
cout.tie(NULL);  
memset(ans, 0, sizeof(ans));  
memset(trie, -1, sizeof(trie));  
cin >> s;  
int root = 0, p, cur = 1;  
deep[0] = 0;  
N = s.size();  
for (int i = 0; i < s.size(); ++i)  
{  
p = root;  
for (int j = i; j < s.size(); ++j)  
{  
if (trie[p][s[j] - 'a'] == -1)  
{  
trie[p][s[j] - 'a'] = cur++;  
deep[trie[p][s[j] - 'a']] = deep[p] + 1;  
fa[trie[p][s[j] - 'a']] = p;  
p = trie[p][s[j] - 'a'];  
}  
else  
p = trie[p][s[j] - 'a'];  
}  
last[i + 1] = p;  
}  
memset(matrix,0,sizeof(matrix));  
for (int i = 0; i < s.size(); ++i)  
ch[i + 1] = s[i];  
for (int i = 1; i <= s.size(); ++i)  
dfs1(i, 0), dfs2(i, 0);  
cin >> K;  
for (int i = s.size(); i >0; --i)  
{  
int f = last[i];  
while (f != 0)  
{  
//cout << i << " " << matrix[i][deep[f]+i-1] <<" "<<deep[f]+i-1<< endl;;  
if (matrix[i][deep[f] + i - 1])  
ans[f]++;  
f = fa[f];  
}  
}  
dfs3(0);  
for (list<int>::iterator s=st.begin();s!=st.end();++s)  
cout << char(\*s+'a');  
cout << endl;  
return 0;  
}