```cpp
#include "../bits/stdc++.h"
// 入力文字列に対してマッチするパターンを検索
// O(N + M)
// N := 入力文字列, M := パターン文字列の長さ合計
/*
1. パターンから Trie 木を作成.
2. 各ノードが表す文字列の末尾と一致するノードの内, 文字列長が最大のノードへ辺を張る.
   そのようなノードが存在しなければ根に辺を張る. この処理は bfs で可能.
   各ノードはその祖先が一致する文字列の情報も持つ.
3. PMA(パターンマッチングオートマトン)の完成！
*/
// https://github.com/Suikaba/procon-lib/blob/master/string/aho_corasick.hpp
// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3382323
// 出現文字
struct characters
{
    // lower alphabets
    static int const size = 26;
    static int convert(char c)
    {
        assert('a' <= c && c <= 'z');
        return c - 'a';
    }
    static char invert(int i)
    {
        assert(0 <= i && i < size);
        return 'a' + i;
    }
};

template <typename Characters>
class AhoCorasick
{
    static constexpr int invalidIndex = -1;

    struct PMA
    {
        int fail;
        std::vector<int> next, accept;

        PMA() : fail(invalidIndex), next(Characters::size, invalidIndex) {}
    };

    const int K;
    std::vector<std::unique_ptr<PMA>> nodes;

    int transition(int nodeIndex, char cc)
    {
        assert(0 <= nodeIndex && nodeIndex < (int)nodes.size());
        int c = Characters::convert(cc);
        int now = nodeIndex;
        while (nodes[now]->next[c] == invalidIndex && now != 0)
        {
            now = nodes[now]->fail;
        }
        now = nodes[now]->next[c];
        if (now == invalidIndex)
            now = 0;
        return now;
    }

  public:
    AhoCorasick(const std::vector<std::string> &ts) : K((int)ts.size())
    {
        const int rootIndex = 0;
        // root node
        nodes.push_back(std::make_unique<PMA>());
        nodes[rootIndex]->fail = rootIndex;
        for (int i = 0; i < K; i++)
        {
            int now = rootIndex;
            for (auto cs : ts[i])
            {
                int c = Characters::convert(cs);
                if (nodes[now]->next[c] == invalidIndex)
                {
                    nodes[now]->next[c] = (int)nodes.size();
                    nodes.push_back(std::make_unique<PMA>());
                }
                now = nodes[now]->next[c];
            }
            nodes[now]->accept.push_back(i);
        }

        std::queue<int> que;
        for (int c = 0; c < Characters::size; c++)
        {
            if (nodes[rootIndex]->next[c] != invalidIndex)
            {
                nodes[nodes[rootIndex]->next[c]]->fail = rootIndex;
                que.push(nodes[rootIndex]->next[c]);
            }
        }
        while (!que.empty())
        {
```

```
            int now = que.front();
            que.pop();
            for (int c = 0; c < Characters::size; c++)
            {
                if (nodes[now]->next[c] != invalidIndex)
                {
                    que.push(nodes[now]->next[c]);
                    int nxt = transition(nodes[now]->fail, Characters::invert(c));
                    nodes[nodes[now]->next[c]]->fail = nxt;
                    for (auto ac : nodes[nxt]->accept)
                    {
                        nodes[nodes[now]->next[c]]->accept.push_back(ac);
                    }
                }
            }
        }
    }
    std::vector<std::vector<int>> match(const std::string &str)
    {
        std::vector<std::vector<int>> ret(K);
        int now = 0;
        for (int i = 0; i < (int)str.size(); i++)
        {
            now = transition(now, str[i]);
            for (auto k : nodes[now]->accept)
            {
                ret[k].push_back(i);
            }
        }
        return ret;
    }
};

```