```cpp
#ifndef AHO_CORASICK_H
#define AHO_CORASICK_H

#include <iostream>
#include <iterator>
#include <limits>
#include <list>
#include <string>
#include <sstream>
#include <unordered_set>
#include <vector>

/* Aho-Corasick algorithm for finding uses of previous regular definitions in
 * subsequent regular definitons.
 *
 * Alfred V. Aho and Margaret J. Corasick. 1975. Efficient string matching: an
 * aid to bibliographic search. Commun. ACM 18, 6 (June 1975), 333-340.
 */
class AhoCorasick {
private:
  const unsigned int fail = std::numeric_limits<unsigned int>::max();
  const unsigned int ascii_size = 128u;

  unsigned int newstate;                                  // new state counter
  std::list<std::string> K;                               // list of keywords yi
  std::vector<std::vector<unsigned int>> g;               // goto fn g(s,a)
  std::vector<unsigned int> f;                            // failure fn f(s)
  std::vector<std::unordered_set<std::string>> output;// output fn output(s)
  std::vector<std::vector<unsigned int>> delta;           // next move fn delta(s,a)

  // helper fn: pretty print std::unordered_set<std::string>
  std::string pretty_print(const std::unordered_set<std::string> &S) {
    std::ostringstream oss{};
    auto b = S.begin(), e = S.end();
    oss << "{";
    for (auto i = b; i != e; ++i) oss << (i != b ? ", " : "") << *i;
    oss << "}";
    return oss.str();
  }

  // enter: inserts into the goto graph a path that spells out y
  // input  - keyword y = {a1a2...am}
  //        - newstate counter
  // output - partially computed goto function g(s,a)
  void enter(const std::string& y) {
    auto m = y.size();
    auto state = 0u, j = 0u;
    while (g[state][y[j]] != fail) {
      state = g[state][y[j++]];
    }
    for (auto p = j; p < m; ++p) {
      g[state][y[p]] = ++newstate;
      state = newstate;
      g.push_back(std::vector<unsigned int>(ascii_size, fail));
      output.push_back(std::unordered_set<std::string>{});
```

```
      f.push_back(0u);
      delta.push_back(std::vector<unsigned int>(ascii_size,0u));
    }
    output[state].insert(y);
  }

  // algorithm2: construction of the goto function, g(s,a)
  // input  - set of keywords K = {y1, y2, ..., yk}
  // output - goto function g(s,a)
  //      - partially computed output function output(s)
  void algorithm2() {
    newstate = 0u;
    for (auto y : K) enter(y);
    for (auto& a : g[0])
      if (a == fail)
        a = 0;
  }

  // algorithm3: construction of the failure function, f(s)
  // input  - goto function g(s,a)
  //      - partially computed output function output(s)
  // output - failure function g(s)
  //      - output function output(s)
  void algorithm3() {
    std::list<unsigned int> queue{};
    for (auto a = 0u; a < ascii_size; ++a) {
      auto s = g[0][a];
      if (s != 0u) {
        queue.push_back(s);
        f[s] = 0u;
      }
    }
    while (!queue.empty()) {
      auto r = queue.front();
      queue.pop_front();
      for (auto a = 0u; a < ascii_size; ++a) {
        auto s = g[r][a];
        if (s != fail) {
          queue.push_back(s);
          auto state = f[r];
          while (g[state][a] == fail) {
            state = f[state];
          }
          f[s] = g[state][a];
          auto b = output[f[s]].begin(), e = output[f[s]].end();
          output[s].insert(b, e);
        }
      }
    }
  }

  // algorithm4: construction of DFA, delta(s,a)
  // input  - goto function g(s,a)
  //      - failure function f(s)
  // output - next move function delta(s,a)
```

```
  void algorithm4() {
    std::list<unsigned int> queue{};
    for (auto a = 0u; a < ascii_size; ++a) {
      delta[0][a] = g[0][a];
      if (g[0][a] != 0u) {
        queue.push_back(g[0][a]);
      }
    }
    while (!queue.empty()) {
      auto r = queue.front();
      queue.pop_front();
      for (auto a = 0u; a < ascii_size; ++a) {
        auto s = g[r][a];
        if (s != fail) {
          queue.push_back(s);
          delta[r][a] = s;
        } else {
          delta[r][a] = delta[f[r]][a];
        }
      }
    }
  }

public:
  // ctor:
  // input  - list of keywords k = {y1, y2, ..., yk}
  AhoCorasick(const std::list<std::string>& k) : newstate(0u), K(k) {
    g.push_back(std::vector<unsigned int>(ascii_size, fail)); // init goto fn
    output.push_back(std::unordered_set<std::string>{});      // init output fn
    f.push_back(0u);                                          // init failure fn
    delta.push_back(std::vector<unsigned int>(ascii_size,0u));// init next mv fn
    algorithm2();    // compute goto fn, partially compute output fn
    algorithm3();    // compute failure fn, and finish computing output fn
    algorithm4();    // compute next move fn (DFA)
  }

  const std::list<std::string>& Keywords() { return K; }
  const std::vector<std::vector<unsigned int>>& Goto() { return g; }
  const std::vector<unsigned int>& Failure() { return f; }
  const std::vector<std::unordered_set<std::string>>& Output() {return output;}
  const std::vector<std::vector<unsigned int>>& Delta() { return delta; }

  // default dtor
  ~AhoCorasick() = default;

  // algorithm1: pattern matching machine (M)
  // input  - text string x = a1a2...an
  //      - goto function g(s,a)
  //      - failure function f(s)
  //      - output function output(s)
  // output - locations at which keywords occur in x (stdout)
  void algorithm1(const std::string& x) {
    unsigned int state{0u};
    auto b = x.begin(), e = x.end();
    for (auto i = b; i != e; ++i) {
```

```
      auto a = *i;
      // while (g[state][a] == fail) { state = f[state]; }
      // state = g[state][a];
      state = delta[state][a];
      if (!output[state].empty()) {
        std::cout << std::distance(b,i) << " : ";
        std::cout << pretty_print(output[state]) << std::endl;
      }
    }
  }
};

#endif // AHO_CORASICK_H
```