Contents

Header

```cpp
typedef uint8_t u8;
typedef uint16_t u16;
typedef uint32_t u32;
typedef uint64_t u64;

typedef int8_t i8;
typedef int16_t i16;
typedef int32_t i32;
typedef int64_t i64;

typedef float f32;
typedef double f64;
typedef long double f80;

#define pb push_back
#define pf push_front
#define fst first
#define snd second
```

Mathematics

Number theory

Given a, b, finds $g = \gcd(a, b)$ and u, v such that $ua + vb = g$
Time: $\mathcal{O}(\log ab)$

```cpp
#include "../header.h"

array<i64, 3> extended_euclid(i64 a, i64 b) {
  if (b == 0)
    return {a, 1, 0};
  auto [g, x, y] = extended_euclid(b, a % b);
  return {g, y, x - y * (a / b)};
}
```

Finds $x^{-1} \bmod m$ in $\mathcal{O}(\log m)$.

```cpp
optional<i64> inv(i64 x, i64 m) {
  auto [g, y, _] = extended_euclid(x, m);
  if (g != 1)
    return {};
  return (y >= 0 ? y % m : m - (-y) % m);
}
```

String algorithms

Aho-Corasick

Builds the Aho-Corasick automaton.
Time: $\mathcal{O}(N)$ where N is the total length of the strings.
Memory: $\mathcal{O}(\Sigma N)$ where Σ is the size of the alphabet.

```cpp
template<int K = 26> class AhoCorasick {
  struct Node {
    Node* tr[K];       // transitions
    Node* suff;        // dictionary suffix
    vector<Node*> adj; // incoming dict suffixes

    Node() : suff(nullptr) {
      fill(tr, tr + K, nullptr);
    }
  };

  Node* root;
  vector<Node*> dict;

  Node* insert(const string &s) {
    Node* curr = root;
    for (auto c: s) {
      if (!curr->tr[c - 'a'])
        curr->tr[c - 'a'] = new Node;
      curr = curr->tr[c - 'a'];
    }

    return curr;
  }

  void get_suffixes() {
    queue<Node*> q;

    for (int i = 0; i < K; i++) {
      if (root->tr[i]) {
        root->tr[i]->suff = root;
        root->adj.push_back(root->tr[i]);
        q.push(root->tr[i]);
      } else {
        root->tr[i] = root;
      }
    }

    while (!q.empty()) {
      Node* curr = q.front(); q.pop();

      for (int i = 0; i < K; i++) {
        if (curr->tr[i]) {
          curr->tr[i]->suff = curr->suff->tr[i];
          curr->tr[i]->suff->adj
            .push_back(curr->tr[i]);
          q.push(curr->tr[i]);
        } else {
          curr->tr[i] = curr->suff->tr[i];
        }
      }
    }
  }

public:

  AhoCorasick(const vector<string> &words) {
    root = new Node;
    for (auto &word: words) {
      dict.push_back(insert(word));
    }
    get_suffixes();
  }
};
```