Contents

1 Data structures

1.1 Fenwick tree

```
vector<int> data(n);

// Adds value val on position pos
auto addval = [&](int pos, int val) {
    while (pos < n) {
        data[pos] += val;
        pos |= (pos + 1);
    }
};

// Returns sum of values on half-interval [0, pos)
auto getsum = [&](int pos) {
    int ret = 0;
    while (pos) {
        ret += data[pos - 1];
        pos = pos & (pos - 1);
    }
    return ret;
};
```

Warning: No tests

2 Graphs

2.1 Hungarian algorithm

```
namespace hungary {
    const int N = 210;

    int a[N][N];
    int ans[N];

    int calc(int n, int m) {
        ++n, ++m;
        vector<int> u(n), v(m), p(m), prev(m);
        for (int i = 1; i < n; ++i) {
            p[0] = i;
            int x = 0;
            vector<int> mn(m, INF);
            vector<int> was(m, 0);
            while (p[x]) {
                was[x] = 1;
                int ii = p[x], dd = INF, y = 0;
                for (int j = 1; j < m; ++j) if (!was[j]) {
                    int cur = a[ii][j] - u[ii] - v[j];
                    if (cur < mn[j]) mn[j] = cur, prev[j] = x;
                    if (mn[j] < dd) dd = mn[j], y = j;
                }
                for (int j = 0; j < m; ++j) {
                    if (was[j]) u[p[j]] += dd, v[j] -= dd;
                    else mn[j] -= dd;
                }
                x = y;
```

```
            }
            while (x) {
                int y = prev[x];
                p[x] = p[y];
                x = y;
            }
        }
        for (int j = 1; j < m; ++j) {
            ans[p[j]] = j;
        }
        return -v[0];
    }
    // How to use:
    // * Set values to a[1..n][1..m] (n <= m)
    // * Run calc(n, m) to find minimum
    // * Optimal edges are (i, ans[i]) for i = 1..n
    // * Everything works on negative numbers
    //
    // !!! I don't understand this code, it's copypasted from e-maxx
}
```

Warning: No tests

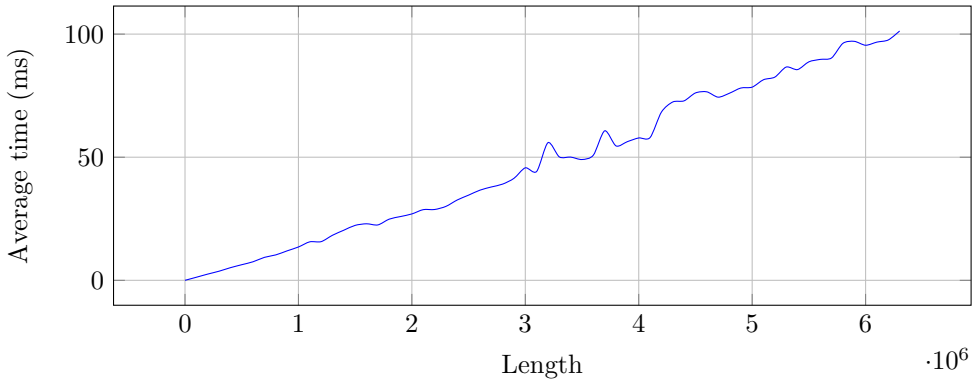
3 String algorithms

3.1 Manacher algorithm (longest palindrome for every center)

```
// returns vector ret of length (|s| * 2 - 1),
//   ret[i * 2] -- maximal length of palindrome with center in i-th symbol
//   ret[i * 2 + 1] -- maximal length of palindrome with center between i-th and (i + 1)-th symbols
vector<int> find_palindromes(string const& s) {
    string t(szof(s) * 2 - 1, '$');
    for (int i = 0; i < szof(s); ++i) {
        t[i * 2] = s[i];
    }

    int c = 0, r = 1;
    vector<int> d(szof(t));
    d[0] = 1;
    for (int i = 1; i < szof(t); ++i) {
        if (i < c + r) {
            d[i] = min(c + r - i, d[2 * c - i]);
        }
        while (i - d[i] >= 0 && i + d[i] < szof(t) && t[i - d[i]] == t[i + d[i]]) {
            ++d[i];
        }
        if (i + d[i] > c + r) {
            c = i;
            r = d[i];
        }
    }

    for (int i = 0; i < szof(t); ++i) {
        if (i % 2 == 0) {
            d[i] -= 1 - (d[i] & 1);
        } else {
            d[i] &= ~1;
```

```
        }
    }

    return d;
}
```

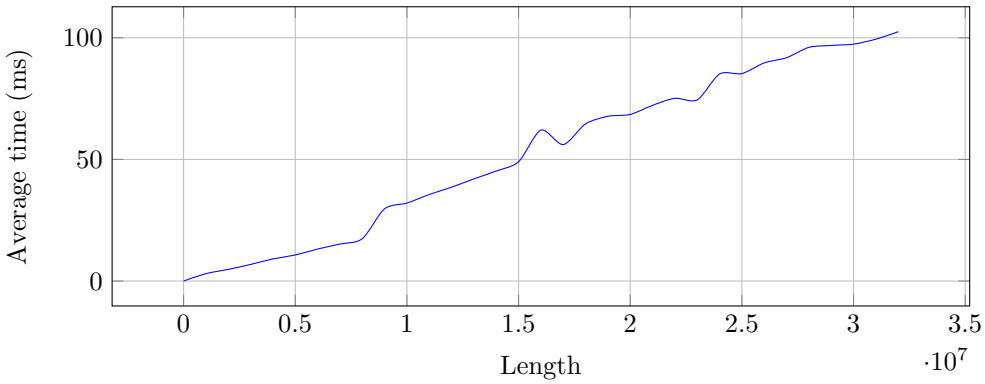
✓Tests passed



3.2 Prefix function

```
vector<int> calc_pref(string s) {
    vector<int> pref(szof(s));
    for (int i = 1; i < szof(s); ++i) {
        pref[i] = pref[i - 1];
        while (pref[i] && s[pref[i]] != s[i]) {
            pref[i] = pref[pref[i] - 1];
        }
        if (s[pref[i]] == s[i]) {
            ++pref[i];
        }
    }
    return pref;
}
```

✓Tests passed



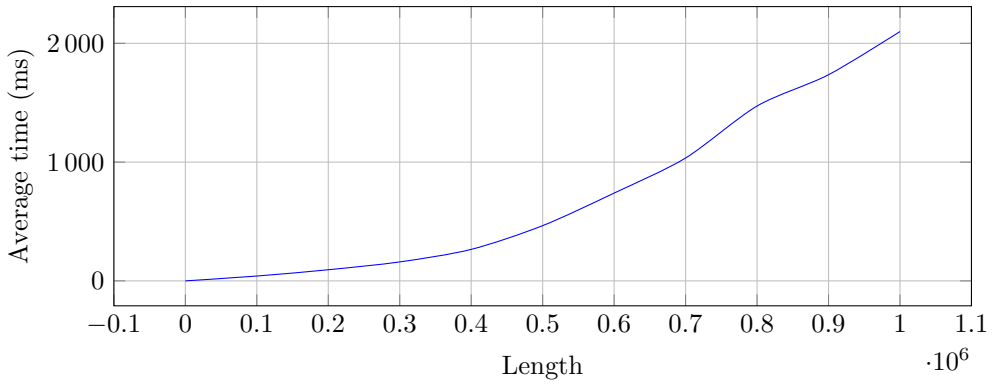
3.3 Suffix structures

3.3.1 Suffix array

```
vector<int> suff_array(string s) {
    s += '\0';
```

```
    int n = s.size();
    vector<int> classes(s.begin(), s.end()), new_classes(n);
    vector<int> order(n), new_order(n);
    iota(order.begin(), order.end(), 0);

    function<int(int)> mod = [&](int num) {
        if (num >= n) {
            return num - n;
        }
        return num;
    };

    vector<int> from(max(n, 128) + 1);
    for (int num : classes) {
        from[num + 1]++;
    }

    for (int i = 1; i < (int) from.size(); ++i) {
        from[i] += from[i - 1];
    }

    for (int i = 0; i < n; i == 0 ? i = 1 : i <<= 1) {
        for (int j = 0; j < n; ++j) {
            int pos = mod(order[j] - i + s.size());
            new_order[from[classes[pos]]++] = pos;
        }

        swap(order, new_order);

        int cnt = -1;
        for (int j = 0; j < n; ++j) {
            if (j == 0 || classes[order[j]] != classes[order[j - 1]] || classes[mod(order[j] + i)]
            ↪  != classes[mod(order[j - 1] + i)]) {
                ++cnt;
                from[cnt] = j;
            }
            new_classes[order[j]] = cnt;
        }

        swap(classes, new_classes);
    }

    order.erase(order.begin());
    return order;
}
```

✓Tests passed



4 FFT

4.1 FFT by modulo

```
namespace fft {
    int const BP = 20, SZ = 1 << BP;
    int const INV_SZ = mpow(SZ, MOD - 2);
    int perm[SZ], roots[SZ];
    int arr1[SZ], arr2[SZ];

    void fft(int* arr) {
        for (int i = 0; i < SZ; ++i) {
            if (perm[i] > i) {
                swap(arr[i], arr[perm[i]]);
            }
        }

        for (int i = 1, diff_pow = SZ >> 1; i < SZ; i <<= 1, diff_pow >>= 1) {
            for (int j = 0; j < SZ; j += i * 2) {
                int cur_pow = 0;
                for (int k = 0; k < i; ++k) {
                    int b = mult(arr[j + i + k], roots[cur_pow]);
                    arr[j + i + k] = sum(arr[j + k], MOD - b);
                    add(arr[j + k], b);
                    cur_pow += diff_pow;
                }
            }
        }
    }

    void fill_arr(vector<int> const& a, int* arr) {
        for (int i = 0; i < SZ; ++i) {
            if (i < (int) a.size()) {
                arr[i] = a[i];
            } else {
                arr[i] = 0;
            }
        }
    }

    vector<int> mult(vector<int> const& a, vector<int> const& b) {
        fill_arr(a, arr1);
        fft(arr1);
        fill_arr(b, arr2);
```

```
        fft(arr2);
        for (int i = 0; i < SZ; ++i) {
            arr1[i] = mult(arr1[i], arr2[i]);
        }
        fft(arr1);
        reverse(arr1 + 1, arr1 + SZ);
        vector<int> ret;
        for (int i = 0; i < SZ; ++i) {
            ret.push_back(mult(arr1[i], INV_SZ));
        }
        while (ret.back() == 0) {
            ret.pop_back();
        }
        return ret;
    }

    void init() {
        int rt = 646; // this is precalculated 2^20-th root of 1 for MOD = 998244353
        /*
        for (int i = 0; i < MOD; ++i) {
            if (mpow(i, 1 << (BP - 1)) == MOD - 1) {
                rt = i;
                break;
            }
        }
        */

        roots[0] = 1;
        for (int i = 1; i < SZ; ++i) {
            perm[i] = (perm[i >> 1] >> 1) | ((i & 1) << (BP - 1));
            roots[i] = mult(roots[i - 1], rt);
        }
    }
}
```

Error: Compilation error while compiling test

4.2 FFT in complex numbers

Warning: Leaf directory without any information

5 Convex hull trick

5.1 Arbitrary order of lines

```
// Adds line k * x + b
void add_line(ll k, ll b) {

}
```

Warning: No tests