

# Team Notebook



## template

```
#include <bits/stdc++.h>
using namespace std;

template <typename T>
using vec = vector<T>;
using ll = long long;
#define all(x) x.begin(), x.end()

int main() {
    cin.tie(0) -> sync_with_stdio(0);
    return 0;
}
```

## Data Structures

### hashmap

**description:** Hash map with mostly the same API as `unordered_map`, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

struct chash {
    const uint64_t C = ll(4e18 * acos(0)) | 71;
    ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
gp_hash_table<ll, int, chash> h({}, {}, {}, {}, {1<<16});
```

### order\_statistic\_tree

**time:**  $O(\log n)$

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template<class T>
```

```
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
```

### union\_find

**time:**  $O(\alpha(n))$

```
struct union_find {
    vec<int> e;
    union_find(int n) : e(n, -1) {}
    bool sameSet(int a, int b) {
        return find(a) == find(b);
    }
    int size(int x) {
        return -e[find(x)];
    }
    int find(int x) {
        return e[x] < 0 ? x : e[x] = find(e[x]);
    }
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        e[a] += e[b]; e[b] = a;
        return true;
    }
};
```

### convex\_hull\_trick

**description:** Container where you can add lines of the form  $kx + m$ , and query maximum values at points  $x$ .

**time:**  $O(\log n)$

```
struct line {
    mutable ll k, m, p;
    bool operator<(const line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};
```

```
struct line_container : multiset<line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
```

```

    return a / b - ((a ^ b) < 0 && a % b);
}
bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
}
void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
        isect(x, erase(y));
}
ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
}
};

```

---

```

    if (p[i] == pat.size())
        res.push_back(i - 2 * pat.size());
}
return res;
}

```

---

## Strings

### kmp

**description:**  $\text{pi}[x]$  computes the length of the longest prefix of  $s$  that ends at  $x$ , other than  $s[0 \dots x]$  itself (abacaba  $\rightarrow$  0010123). Can be used to find all occurrences of a string.

**time:**  $O(n)$

```

vec<int> pi(const string& s) {
    vec<int> p(s.size());
    for (int i = 0; i < s.size(); i++) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}

vec<int> match(const string& s, const string& pat) {
    vec<int> p = pi(pat + '\0' + s), res;
    for (int i = 0; i < p.size(); i += p.size()-s.size()) {

```