

## 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>;
```

---

## strings

### kmp

**description:** `pi[x]` computes the length of the longest prefix of `s` that ends at `x`, other than `s[0...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()) {
        if (p[i] == pat.size())
            res.push_back(i - 2 * pat.size());
    }
    return res;
}
```

---