LU_dAREdevils

Leading University

- **FastIO**: `ios::sync_with_stdio(false); cin.tie(0);`
- **File Handeling:**

```
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
freopen("error.txt", "w", stderr);
auto st = clock(); // Current time should be placed on the first line
cerr << "Time = " << 1.0 * (clock() - st) / CLOCKS_PER_SEC << "\n";
#endif
```

- **next_permutation():** It is used to rearrange the elements in the range [first, last) into the next lexicographically greater permutation. {{1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}};

```
int arr[] = {1, 2, 3};                    => O(n*n!)
do{
    //Add any conditions;
   cout << arr[0] << " " << arr[1] << " " << arr[2] << "\n";
} while (next_permutation(arr, arr + 3));
```

- Erase Duplicate value in sorted vector**: v.erase(unique(v.begin(), v.end()), v.end());**
- **Better then rand()** function:

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count()); // mt19937_64 (long long)
auto my_rand(long long l, long long r)  { // random value generator [l, r]
  return uniform_int_distribution<long long>(l, r)(rng);
}
```

- **merge():** Merge two sorted arrays using merge present algorithm header file. **The Arrays must be sorted.**
  => O(vec1.size() + vec2.size() **)**

```
merge(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(), back_inserter(finalVec));
merge(st1.begin(), st1.end(), st2.begin(), st2.end(), inserter(st[node], st.begin()));
```

- **Policy based DS**: The complexity of the **insert** and **erase** functions is **O(log n).**

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <typename T> using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
template <typename T, typename R> using ordered_map = tree<T, R, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;

// *s.find_by_order(k): K-th element in a set (counting from zero).
// s.order_of_key(k): Number of items strictly smaller than k. (same as, lower_bound of k)
// less_equal<T> => for ordered_multiset or, ordered_multimap.
ordered_set<int> s;  ordered_map<int, ll>mp;  // we can change the data type.
```

❖ **gp_hash_table<int, int>:** Same as unordered_map, but **faster** than unordered_map.

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        x += FIXED_RANDOM;
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
```