Disjoint Sets

Disjoint sets es una estructura de datos que permite, muy rápidamente, pegar elementos. Tiene heurística de compresión.

- Tiempo para merge y FindRoot: Amortizado $O(\log^*(n))$

```
#include <algorithm>
#include <iostream>
#include <numeric>
#include <vector>

class disjoint_sets
{
public:
    using size_type = std::int64_t;
    using index_type = std::int64_t;
    explicit disjoint_sets(index_type n) : parent(n), m_num_components(n)
    {
        std::iota(parent.begin(), parent.end(), 0L);
    }

    index_type find_root(index_type t)
    {
        std::vector<index_type> branch;
        branch.emplace_back(t);
        while (t != parent[t])
        {
            t = parent[t];
            branch.emplace_back(t);
        }
        for (auto u : branch)
            parent[u] = t;
        return t;
    }

    void reset()
    {
        std::iota(parent.begin(), parent.end(), 0);
        m_num_components = size();
    }

    void merge(index_type a, index_type b)
    {
        index_type ra = find_root(a);
```

```
        index_type rb = set_parent(b, ra);
        if (ra != rb)
        {
            --m_num_components;
        }
    }

    bool are_in_same_connected_component(index_type a, index_type b)
    {
        return find_root(a) == find_root(b);
    }

    size_type num_components() const { return m_num_components; }

    index_type size() const { return parent.size(); }

    auto& parents() const { return parent; }

private:
    // returns ORIGINAL parent of x
    index_type set_parent(index_type x, index_type p)
    {
        while (x != parent[x])
        {
            index_type t = parent[x];
            parent[x] = p;
            x = t;
        }
        parent[x] = p;
        return x;
    }

    std::vector<index_type> parent;
    size_type m_num_components;
};

int main()
{
    disjoint_sets D(4);

    std::cout << "Num components: " << D.num_components() << std::endl;
    D.merge(0, 1);
    std::cout << "Num components: " << D.num_components() << std::endl;
    D.merge(2, 3);
    std::cout << "Num components: " << D.num_components() << std::endl;
```

```
    D.merge(0, 3);
    std::cout << "Num components: " << D.num_components() << std::endl;
    D.merge(1, 2);
    std::cout << "Num components: " << D.num_components() << std::endl;

    return 0;
}
```

Output:

```
Num components: 4
Num components: 3
Num components: 2
Num components: 1
Num components: 1
```