Lowest Common Ancestors

Una clase que, dado un árbol, puede responder a la pregunta "¿quién es el ancestro común más cercano de dos vértices u y v?" rápidamente.

Se incluyen sólo la implementación de los 2^i-ancestros. Hay una mejor pero más complicada de escribir.

- Tiempo de preprocesamiento: $O(n \log(n))$.
- Tiempo para pregunta: $O(\log(n))$

REQUIERE: Graph, Tree

```
#include <stack>

#include "Graph.hpp"
#include "Tree.hpp"

using Vertex = Graph::Vertex;

class LCA
{
public:
    using Vertex = Graph::Vertex;
    LCA(const Graph& G, Vertex root)
        : L(height_map(G, root))
        , A(G.num_vertices(),
            std::vector<Vertex>(
              std::log2(*std::max_element(L.begin(), L.end()) + 1) + 1, -1))
    {
        auto parents = set_root(G, root);

        // The 2^0-th ancestor of v is simply the parent of v
        for (auto v : G.vertices())
            A[v][0] = parents[v];

        for (int i = 1; i < log_height(); ++i)
        {
            for (auto v : G.vertices())
            {
                // My 2^i-th ancestor is the 2^{i-1} ancestor of my 2^{i-1}
                // ancestor!
                if (A[v][i - 1] != -1)
                    A[v][i] = A[A[v][i - 1]][i - 1];
            }
        }
```

```
    }

    Vertex FindLCA(Vertex u, Vertex v) const
    {
        if (L[u] < L[v])
            std::swap(u, v);

        u = AncestorAtLevel(u, L[v]);

        if (u == v)
            return u;

        for (int i = std::log2(L[u]); i >= 0; --i)
        {
            if (A[u][i] != -1 && A[u][i] != A[v][i])
            {
                u = A[u][i];
                v = A[v][i];
            }
        }

        return A[u][0];
    }

    const std::vector<std::vector<Vertex>>& Ancestors() const { return A; }
    const auto& Level() const { return L; }

private:
    // L[v] is the level of vertex v
    std::vector<int> L;

    // A[v][i] is the 2^i ancestor of vertex v
    std::vector<std::vector<Vertex>> A;

    int log_height() const { return A[0].size(); }

    Vertex AncestorAtLevel(Vertex u, int lvl) const
    {
        int d = L[u] - lvl;
        while (d > 0)
        {
            int h = std::log2(d);
            u = A[u][h];
            d -= (1 << h);
        }
```

```
        return u;
    }
};

using namespace std;

template <class T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& A)
{
    for (const auto& x : A)
        os << x << ' ';
    return os;
}

int main()
{
    Graph tree(5);
    tree.add_edge(1, 0);
    tree.add_edge(1, 2);
    tree.add_edge(2, 3);
    tree.add_edge(2, 4);

    LCA lca(tree, 1);

    cout << "LCA of 0 and 4: " << lca.FindLCA(0, 4) << endl;
    cout << "LCA of 3 and 4: " << lca.FindLCA(3, 4) << endl;

    return 0;
}
```

Output:

```
LCA of 0 and 4: 1
LCA of 3 and 4: 2
```