```
std::vector<int>
        leaveBiggestComponent(std::vector<ImageFeatures> &features, std::vector<MatchesInfo>
&pairwise_matches,
                              float conf_threshold) {
    const int num_images = static_cast<int>(features.size());

    DisjointSets comps(num_images);
    for (int i = 0; i < num_images; ++i) {
        for (int j = 0; j < num_images; ++j) {
            m21log(math21_string_to_string(i, j, pairwise_matches[i * num_images +
j].confidence));
            if (pairwise_matches[i * num_images + j].confidence < conf_threshold) {
                continue;
            }
            int comp1 = comps.findSetByElem(i);
            int comp2 = comps.findSetByElem(j);
            if (comp1 != comp2)
                comps.mergeSets(comp1, comp2);
        }
    }

    int max_comp = static_cast<int>(std::max_element(comps.size.begin(),
comps.size.end()) -
                                    comps.size.begin());

    std::vector<int> indices;
    std::vector<int> indices_removed;
    for (int i = 0; i < num_images; ++i)
        if (comps.findSetByElem(i) == max_comp)
            indices.push_back(i);
        else
            indices_removed.push_back(i);

    std::vector<ImageFeatures> features_subset;
    std::vector<MatchesInfo> pairwise_matches_subset;
    for (size_t i = 0; i < indices.size(); ++i) {
        features_subset.push_back(features[indices[i]]);
        for (size_t j = 0; j < indices.size(); ++j) {
            pairwise_matches_subset.push_back(pairwise_matches[indices[i] *
num_images + indices[j]]);
            pairwise_matches_subset.back().src_img_idx = static_cast<int>(i);
```

```
                        pairwise_matches_subset.back().dst_img_idx = static_cast<int>(j);
                    }
                }

                if (static_cast<int>(features_subset.size()) == num_images)
                    return indices;

                LOG("Removed some images, because can't match them or there are too similar images: (");
                LOG(indices_removed[0] + 1);
                for (size_t i = 1; i < indices_removed.size(); ++i)
                    LOG(", " << indices_removed[i] + 1);
                LOGLN(").");
                LOGLN("Try to decrease the match confidence threshold and/or check if you're stitching duplicates.");

                features = features_subset;
                pairwise_matches = pairwise_matches_subset;

                return indices;
            }
class DisjointSets {
            public:
                DisjointSets(int elem_count = 0) { createOneElemSets(elem_count); }

                void createOneElemSets(int elem_count);

                int findSetByElem(int elem);

                int mergeSets(int set1, int set2);

                std::vector<int> parent;
                std::vector<int> size;

            private:
                std::vector<int> rank_;
            };
void DisjointSets::createOneElemSets(int n) {
                rank_.assign(n, 0);
                size.assign(n, 1);
                parent.resize(n);
```

```
    for (int i = 0; i < n; ++i)
        parent[i] = i;
}


int DisjointSets::findSetByElem(int elem) {
    int set = elem;
    while (set != parent[set])
        set = parent[set];
    int next;
    while (elem != parent[elem]) {
        next = parent[elem];
        parent[elem] = set;
        elem = next;
    }
    return set;
}


int DisjointSets::mergeSets(int set1, int set2) {
    if (rank_[set1] < rank_[set2]) {
        parent[set1] = set2;
        size[set2] += size[set1];
        return set2;
    }
    if (rank_[set2] < rank_[set1]) {
        parent[set2] = set1;
        size[set1] += size[set2];
        return set1;
    }
    parent[set1] = set2;
    rank_[set2]++;
    size[set2] += size[set1];
    return set2;
}
```