Lab 3

1 实验目的

1. 掌握近似算法的基本设计思想与方法，
2. 掌握集合覆盖问题近似算法的设计思想与方法，
3. 熟练使用高级编程语言实现近似算法，
4. 利用实验测试给出不同近似算法的性能以理解其优缺点。

2 实验问题

集合覆盖问题。

3 实验步骤

3.1 实现基于贪心策略的近似算法

主要思想

总是选择能覆盖最多未被覆盖元素的子集。

```
vector<unordered_set<int>> GreedySetCover(unordered_set<int> universe,
vector<unordered_set<int>> collection) {
    vector<unordered_set<int>> cover;

    while (!universe.empty()) {
        size_t max_size = 0;
        vector<unordered_set<int>>::iterator it = collection.begin(), max_it = it;
        while (it != collection.end()) {
            size_t size = count_if(it->begin(), it->end(), [&](int i) { return
universe.find(i) != universe.end(); });
            if (size > max_size) {
                max_size = size;
                max_it = it;
            }
            ++it;
        }
        cover.push_back(*max_it);
        erase_if(universe, [&](int i) { return max_it->find(i) != max_it->end(); });
        collection.erase(max_it);
    }
    return cover;
}
```

3.2 实现一个基于线性规划近似算法

主要思想

集合覆盖问题可以等价为0-1整数规划，将其松弛成线性规划，利用c++ op-tools库即可求解。

```
vector<unordered_set<int>> ILPSetCover(unordered_set<int> universe,
vector<unordered_set<int>> collection) {
    vector<unordered_set<int>> cover;
    vector<vector<size_t>> coefficients;
    size_t max_frequency = 0;

    for (int e : universe) {
        coefficients.push_back(vector<size_t>());
        for (size_t i = 0; i < collection.size(); ++i) {
            if (collection[i].find(e) != collection[i].end()) {
                coefficients.back().push_back(i);
            }
        }
        max_frequency = max(max_frequency, coefficients.back().size());
    }

    unique_ptr<MPSolver> solver(MPSolver::CreateSolver("GLOP"));

    // Create variables.
    vector<MPVariable*> variables;
    for (size_t i = 0; i < collection.size(); ++i) {
        MPVariable* const x = solver->MakeNumVar(0.0, 1, "");
        variables.push_back(x);
    }

    // Create linear constraints.
    for (size_t i = 0; i < coefficients.size(); ++i) {
        MPConstraint* const ct = solver->MakeRowConstraint(1, MPSolver::infinity());
        for (auto c : coefficients[i]) {
            ct->SetCoefficient(variables[c], 1);
        }
    }

    // Create the objective function.
    MPObjective* const objective = solver->MutableObjective();
    for (auto v : variables) {
        objective->SetCoefficient(v, 1);
    }
    objective->SetMinimization();

    solver->Solve();

    for (size_t i = 0; i < variables.size(); ++i) {
        if (variables[i]->solution_value() >= 1.0 / max_frequency) {
            cover.push_back(collection[i]);
        }
    }

    return cover;
}
```

4 测试算法性能

- N=100

	运行时间	结果大小
Greedy	0.882787ms	36
ILP	0.980678ms	35

- N=1000

	运行时间	结果大小
Greedy	73.6977ms	373
ILP	48.1326ms	365

- N=5000

	运行时间	结果大小
Greedy	1882.13ms	1889
ILP	1045.4ms	1773

从测试结果上看，线性规划法优于贪心法。