Project 3: Minimum Overlap Coverage

Instructions

This project implementation will find the Minimum Overlap of sets that cover all points. For example, suppose we have multiple group projects for our class that needed to be presented at the end of the semester. Your professor must grade every student at least once, but wants to minimize the number grades that must be performed. Your professor does not mind grading multiple projects.

Students are members of multiple projects, but only want to present once. How can we minimize the number of times a student has to present? Everyone in a group has to present (no partial group presentations) and a grade must be assigned for each group member.

To recap, here are the requirements in a list:

- Each student is issued an ID number starting at 0. We do not care about their names.
- Each group consisted of a set of numbers. A group cannot have duplicate students.
- A query consisted of a group of numbers. A student may be in there more than one (your professor misplaced a past grade.)
- When a group presents, all students are graded.
- Your function returns the minimum number of grades your professor has to assess.
- If it is not possible, return `UINT_MAX` (or -1, they are the same.)

Files

- `data/*` – groups of sets that represent the groups that *may* be present.
- `src/main.cpp` – a general-purpose test program for running test cases on your code. It also produces timings.
- `src/MinimumOverlap.hpp` – the class declaration that computes the minimum overlap.
- `src/MinimumOverlap.cpp` – the source file for the class that computes the minimum overlap.
- `data/simple.set` – the example data set provided below.
- `data/small30.set` – small data set for evaluating basic timings.
- `data/medium30.set` – the data set you should strive to master.
- `data/large30.set` – a larger data set over `medium30.set`
- `data/large50.set` – the largest data set I will test.

Set/Group File

Each set/group file contains multiple sets. Each set is contained on one line and is a space-delimited list of numbers. Each number represents a student's ID. For Example:

```
0 2 5
0 2 3
1
2 3 6
1 4 3
5 6
```

Given the above sets, here are few examples of the minimum overlap (`min()`).

Call	Answer	Represented Sets (i.e., Grades/Assessments)
`min("0 2 6")`	5	"0 2 5" and "5 6"
`min("1")`	1	"1"
`min("1 2 2 3")`	7	"0 2 3", "1", and "2 3 6". Note that the input in this example requires a cover with two 2's.
`min("4 4")`	DNE	Stands for "Does Not Exist." Only one set contains a 4.

Interface

```
#ifndef MinimumOverlap_HPP
#define MinimumOverlap_HPP

#include <string>
#include <vector>

class MinimumOverlap {
public:

    // Passes in a string pointing to the set/group file.
    // Make sure you store all the sets/groups!
    MinimumOverlap(const std::string &setFile);

    // Returns the minimum number of overlaps for a cover.
    // If print is true, it prints the set(s).
    unsigned int findMinimumOverlap(const std::vector<int> &cover,
            const bool print) const;
};

#endif
```

Algorithm Setup

- `SOFAR` represents the sets you have chosen.
- `INPUT` represents the student/IDs that still need to be chosen.
- `print` is for printing debug information when `true`.
- `min` represents the minimum number of overlap found so far in a cover.
- `RESULT` is what is leftover from removing the evaluating set.

```
unsigned int findMinimumOverlap (SOFAR, INPUT, bool print) const {

    unsigned int min = UINT_MAX;

    for each set, S, in the set file that is not in SOFAR {
        RESULT = INPUT excluding that values in S
        Add (e.g., append) S to SOFAR
        if (RESULT is empty) { // A cover was found
            if (SOFAR has fewer students selected than min) {
                min = the size of SOFAR.
            }
        }
        else { //didn't find a cover yet, so try with what is left
            // Recursively call with S
            const int cmin = findMinimumOverlap(SOFAR, RESULT, print);
            // If the newly found min is smaller, then update min.
            if (cmin < min) {
                min = cmin;
            }
        }
        REMOVE S from SOFAR so we can evaluate the next S.
    }
    return min;
}
```

As usual, you may not change the public interface but may add any private data and methods. Because we are using recursive backtracking you **must** add at least one additional method for the recursion to work. Keep in mind the above is pseudo-code (I never defined SOFAR's type.) Notice that data stored in the private section SHOULD NOT depend on a particular call to `findMinimumOverlap` but only on the set/group file. Other data should be declared locally within `findMinimumOverlap` and passed as parameters when needed. Constant pass-by-reference is faster than pass-by-value for any data items bigger than a single integer, so consider using that even if you don't change the value.

You may create as many helper classes. The public interface of those classes is completely up to you as well, subject to good design criteria. In particular, you will need to use a (class or struct) to represent a set/group. You may use anything from the STL or roll your own.

Example Runs

Listing 1: The most inefficient solution still solves for small covers quickly.

```
$ ./min-overlap-worst data/simple.set no "1 2 3 4" "0 1 2 3" "0 2 4" \
    "1 2 2 3" "4 0" "4 4"
Arguments were validated, calling constructor...
data/simple.set took 0.000014 seconds preprocessing.
Cover Tested                                    Minimum # of Els  Time (sec)
{1 2 3 4}                                                    6    0.000002
{0 1 2 3}                                                    4    0.000002
{0 2 4}                                                      6    0.000001
{1 2 2 3}                                                    7    0.000002
{4 0}                                                        6    0.000001
{4 4}                                                      DNE    0.000001
Median time to find a cover: 0.000002 seconds.

Timing information written to 'data/simple-timings.dat'
```

Listing 2: Example run for an implementation worth 60% on small30.

```
$ ./min-overlap-60 data/small30.set no "1 2 3 4 5 6 7 8" "0 3 7 10 12 13 14"
    "1 3 4 5 6 9 12" "0 1 2 3 4 5 6 7 8 9 10 11 12 13 14"
Arguments were validated, calling constructor...
data/small30.set took 0.000041 seconds preprocessing.
Cover Tested                                    Minimum # of Els  Time (sec)
{1 2 3 4 5 6 7 8}                                           17    0.001257
{0 3 7 10 12 13 14}                                         16    0.000213
{1 3 4 5 6 9 12}                                            21    0.000661
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14}                        27    0.009630
Median time to find a cover: 0.001257 seconds.

Timing information written to 'data/small30-timings.dat'
```

Listing 3: Example run for an implementation worth 60% on small100.

```
./min-overlap-test data/small100.set no "1 2 3 4 5 6 7 8" "0 3 7 10 12 13
   14" "1 3 4 5 6 9 12" "2 4 6 8 9 12 17 21 23 24" "0 1 2 3 4 5 6 7 8 9 10
   11 12 13 14"
Arguments were validated, calling constructor...
data/small100.set took 0.000082 seconds preprocessing.
Cover Tested                                    Minimum # of Els  Time (sec)
{1 2 3 4 5 6 7 8}                                            9    0.015083
{0 3 7 10 12 13 14}                                          9    0.003823
{1 3 4 5 6 9 12}                                            10    0.009584
{2 4 6 8 9 12 17 21 23 24}                                  13    0.108969
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14}                        16    3.826008
Median time to find a cover: 0.015083 seconds.

Timing information written to 'data/small100-timings.dat'
```

Listing 4: Example run for an implementation worth 100% on large30.

```
$ ./min-overlap-100 data/large30.set no "1 2 3 4 5 6 7 8" "0 3 7 10 12 13
   14" "1 3 4 5 6 9 12" "2 4 6 8 9 12 17 21 23 24" "0 1 2 3 4 5 6 7 8 9 10
   11 12 13 14"
Arguments were validated, calling constructor...
data/large30.set took 0.000052 seconds preprocessing.
Cover Tested                                        Minimum # of Els  Time (sec)
{1 2 3 4 5 6 7 8}                                                21    0.000375
{0 3 7 10 12 13 14}                                              13    0.000117
{1 3 4 5 6 9 12}                                                 18    0.000220
{2 4 6 8 9 12 17 21 23 24}                                       23    0.000628
{0 1 2 3 4 5 6 7 8 9 10 11 12 13 14}                             26    0.001512
Median time to find a cover: 0.000375 seconds.

Timing information written to 'data/large30-timings.dat'
```

Memory Management

Ensure there are no memory leaks in your code by running Valgrind!

STL

You may use anything from the STL.

How to turn in

Turn in via GitHub. Ensure the file(s) are in your directory and then:

```
git add .
git commit -m "Solution to Project 3."
git push
```

Assigned

November 1, 2023

Due Date

November 29, 2023 at 11:59 p.m.

Teamwork

No teamwork, your work must be your own.