Assignment 2

Procedures

Turn in answers to the exercises below on the UAF Blackboard Learn site, under Assignment 2 for this class.

- Your answers should consist of two files: `build.h` and `build.cpp`, from Exercise A. These two files should be **attached** to your submission.
- Send only the above! I do not want things I already have, like the test program.
- If you use a revision control system like git, I'd like a link to your repository, also. This is not required, but highly encouraged!
- I may not look at your homework submission immediately. If you have questions, e-mail me.

Exercises (35 pts total)

Exercise A — Exhaustive Search

Purpose

In this exercise, you will write a package that finds an optimal solution to a problem via exhaustive search.

Background

A river flows from north to south. There are cities on the west bank and cities on the east bank. We wish to build bridges across the river. Each bridge will join a west-bank city with an east-bank city. We wish to choose the bridges we build in such a way as to maximize the tolls we collect from people crossing them.

There are two restrictions on the bridges we build. First, no city can have more than one bridge reaching it. Second, no two bridges can cross each other.

Concerning the second condition: the cities on each bank are numbered. There are *w* cities on the west bank, numbered 0, 1, …, *w*-1 (starting at zero, because we're computer scientists), and there are *e* cities on the east bank, numbered 0, 1, …, *e*-1. Suppose one bridge joins city *a* on the west to city *b* on the east. Another bridge joins city *c* on the west to city *d* on the east. If $a<c$ and $b>d$, then these two bridges cross each other, and we are not allowed to build both.

For each pair of cities we might build a bridge between, we know the toll we will collect from it. Building a bridge never affects the toll collected from another bridge.

Examples

Example 1. Suppose $w=3$ and $e=3$. Each line below describes a bridge we might build; the three numbers are west-bank city, east-bank city, and toll, respectively.

A. $0, 1, 3$.
B. $1, 1, 5$.
C. $1, 2, 4$.
D. $2, 0, 8$.

Answer. We build bridge D, for a total toll of 88.

Note that bridge D crosses every other bridge, so we cannot build a second bridge if we build D.

We could build two bridges: A and C do not cross. But the total toll from that is 3+4=7, so D is a better option.

Example 2. Suppose $w=3$ and $e=3$. Each line below describes a bridge we might build, as before.

A. $0, 1, 3$.
B. $1, 1, 5$.
C. $1, 2, 4$.
D. $2, 0, 8$.
E. $2, 2, 6$.

Answer. We build bridges B and E, for a total toll of 11.

Specification

Each single data item (city index, toll) will be represented by an `int` value.

The description of a bridge will be stored as a `vector<int>` of size `3`.
The `3` items represent, in order:

- The index of the west-bank city (item `0`).
- The index of the east-bank city (item `1`).
- The toll collected (item `3`).

So you might want to do something like this:

```
using Bridge = vector<int>;
```

You are to write a function `build`, prototyped as follows (assuming the above `using`).

```
int build(int w, int e, const vector<Bridge> & bridges);
```

Function `build` takes the number of cities on the west and east banks, respectively, and a `vector` of descriptions of bridges. It returns the maximum total toll that can be collected from a legal set of bridges.

Other requirements:

- Function `build` should be prototyped in file `build.h` and defined in file `build.cpp`, using standard conventions for the structure of header and source files.
- Your code should be in standard C++11/C++14.
- There will be a test program. Your package must compile with the test program, run in a reasonable time (I do not plan on waiting more than a couple of minutes), and pass all tests.
- Your code must use the exhaustive-search algorithmic strategy. You may reduce the size of your search by backtracking when you reach a clear dead end. But other algorithmic cleverness is to be avoided.
- Your code must have no side-effects and produce no output, other than its return value.

Test Program

A test program is available: `build_test.cpp`. If you compile and run this program (unmodified!) with your code, then it will test whether your code works properly.

Do not turn in the test program.

Notes

- You may assume that the input your program is given will be reasonable. The items in the passed `vector` will always have size 3. Given city numbers will always be in range, and tolls will all be positive.
- Your code may be tested with additional input beyond that given in the posted test program.
- Code that passes all the tests will be timed, with the fastest solutions announced in class.

Coding Standards

The following apply to all programming assignments in this class.

- Code must compile & execute using a standard-conforming compiler.
- If a test program is provided, then code must compile with the test program.

The above requirements are absolute; if your code does not compile, then there is no point in turning it in.

In addition, to receive full credit, submitted code should satisfy the following conditions.

- Code should be neat and readable.
- Code should conform to standard conventions (conventional use of header and source files, where applicable, `const`-correctness, etc.).
- Comments should be included, indicating filename, authorship, and last revision date of each file, as well as the purpose of each file and each module that is larger than a function (e.g., a C++ class).
- All comments in the code should be *correct*.
- Programs *may* be based on code written by someone else. However:
  - Your submissions should be largely your own design, and a significant portion of the code should be your own work.
  - You must give credit in the code to the author of any code that is included (in original or modified form) in your submission.
  - Code written by others may only be used in a way that does not violate applicable laws and licenses.