Assignment 5

Assignment 5 is due at **5 pm Friday, November 17**. It is worth 35 points.

Procedures

Turn in answers to the exercises below on the UAF Blackboard Learn site, under Assignment 5 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.
- I may not look at your homework submission immediately. If you have questions, e-mail me.

Exercises (35 pts total)

Exercise A — Dynamic Programming

Purpose

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

Instructions

Redo Assignment 2 Exercise A (the bridge-building problem) with the following changes.

- Your code is to use **dynamic programming** instead of exhaustive search.
  - You may use either a bottom-up or a top-down method.
- **Efficiency expectations** will be greatly increased. A new test program will call your code with larger problems than were used on Assignment 2.

The interface to your code will be identical to that specified in Assignment 2: the same filenames, the same function names & signatures, and the same expected results.

Test Program

A test program is available: `build_test2.cpp`. If you compile and run this program (unmodified!) with your code, then it will test whether your code works properly.

You may also do some of your testing with the test program from Assignment 2: `build_test.cpp`. If you have written your code properly, then the old test program should run *very* quickly—it should probably appear to run instantaneously. You may find the older program useful if you are working on correctness of slow code that is to be optimized later. But you will need to run your code with the later test program eventually.

Notes

- Coding standards are as for Assignment 2.
- There may be more than one way to solve this problem using dynamic programming. Be sure to choose a formulation that results in many overlapping subproblems, so as to obtain the greatest speed-up.
- And as before:
    - 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.