CompSci 260P Spring 2023
Project #0: Getting to Know the Virtual Machine

Due date and time: April 10, 2023 at 6:00 PM.
Remember to read the syllabus to be sure you make yourself eligible to be graded for this assignment.

If you were an undergraduate here and took a class from Professor Thornton, or any class from me that had a programming component, the VM should look very familiar to you. However, this is not the same thing as Project #0 from when you were in that class, so you still have some work to do. Please see the lab manual on the course website for information about setting up the VM.

For students who wish to work in a quiet environment on campus, I recommend loading the VM onto a USB “thumb” drive and taking it to the ICS third floor computer lab. I also recommend you be very careful in backing up your work in progress, no matter where you are doing this, but especially if you are working via a thumb disk in a computer lab. Students who develop their code this quarter on any computer other than an x86 that runs Windows are encouraged to at least run their code on an x86 running Windows; the lab machines offer a great chance to do this. Even though the M1 VM and x86 VM are intended to be identical, and we tried to ensure that they are, we do not guarantee that they are.

The Program’s Requirements

As a warm-up, this project asks you to write and submit two short C++ functions. The functions themselves are not actually the interesting part, though you might find they take you a little bit of time to write. The main goal here is to be sure you’re able to use the VM to do your work, that you learn what you need to know about one of the available text editors to write your program, and that you use the provided tools to get a starting point for your program and submit your code. Even if you normally prefer a different working environment, you would be well-served to use the VM for this project, to be sure that you can use it for your work later in the quarter. Also, if your project does not compile and run in the VM, we will not be able to grade it, and that will cause you to get a zero. Every quarter this happens to some students who, to put it gently, are not pleased with their grade on any such project. Please do not let this happen to you!

See the lab manual for information on how to set up the VM and how to start a project for this class.

For more information about grading, see the relevant section of the lab manual.

The Program’s Requirements

Each of these parts requires you to write a small function in C++. Please use this as an opportunity to gauge your readiness for the class. While solutions to these can easily be found online, doing so would defeat a key benefit you will get from this assignment, to say nothing of the risk you would be taking by submitting work that is not yours.

Take the opportunity to do this project yourself, so that you can check that you can write simple functions in C++, read and understand the requirements documents, and successfully use Gitlab to begin and submit the project.
Part One: TSP
The Traveling Salesperson problem is as follows: we are given a graph with \( n \) vertices. Our goal is to find the smallest-cost Hamiltonian Cycle in the graph. A Hamiltonian Cycle is a cycle that includes every vertex exactly once and returns to the starting point.

For this project, you are given the graph in the form of an adjacency matrix: a 2D vector where element \([i,j]\) is the cost to travel from vertex \( i \) to vertex \( j \). You are guaranteed that every Hamiltonian Cycle has positive cost, and none have cost more than UINT_MAX. All edges have a non-zero cost; the only take that \( graph[i][j] \) will be equal to zero is when \( i=j \). Note that the input graph is directed: it is not guaranteed that \( graph[i][j] == graph[j][i] \).

To accomplish this, you need to implement two functions in tspbruteforce.cpp:

- `std::vector<unsigned> tsp_bruteforce(const std::vector<std::vector<long> > & graph, unsigned n)`
  - Given the graph with the designated number of vertices, return a vector that represents the smallest-cost Hamiltonian Cycle in the graph.
  - In the return value, do not include the starting point twice. That is, the size of the vector you return should be exactly \( n \), and should be a permutation of the integers \( [0, n-1] \)

- `long costOfJourney(const std::vector<std::vector<long> > & graph, const std::vector<unsigned> & journey)`
  - Given the graph and a journey, compute the cost of this permutation for a TSP solution.
  - The last element of the journey vector is not the starting point. Your return value should reflect returning to the starting point as well.
  - You may wish to call this function from tsp_bruteforce.

You may assume all inputs are valid; for example, in costOfJourney, the second parameter will always be a permutation of the unsigned ints in the range of \( [0, n-1] \), where \( n \) is the number of vertices in the given graph. You do not need to do bounds checking or the like.

I encourage you to read the starting point code given in tspbruteforce.cpp and the test cases described in gtest/proj0Test.cpp and ask any questions you may have on EdStem or in office hours.

Part Two: Zeckendorf Representation of Numbers
Recall the Fibonacci numbers: \( F_0 = 0 \), \( F_1 = 1 \), and \( F_n = F_{n-1} + F_{n-2} \) for all \( n > 1 \). The Zeckendorf representation of a positive integer is expressed as the sum of one or more distinct non-consecutive Fibonacci numbers. For example, the number 64 can be expressed as \( 55 + 8 + 1 \). While it is true that 64 can also be written as \( 55 + 5 + 3 + 1 \), that would not be a Zeckendorf representation, as 3 and 5 are consecutive Fibonacci numbers.

Your assignment is to take a non-negative integer and produce its Zeckendorf representation. To do this, you need to implement one function in zeckendorf.cpp:

- `std::bitset<48> zeckendorf(unsigned u)`
  - You are given an unsigned integer; this is a C++ non-negative integer. On our system, this is a 32-bit unsigned integer.
  - Your output is a 48-bit Zeckendorf representation for the same number.
  - No test case will be larger than UINT_MAX for the input parameter, and none will require more than 48 bits for a return value.
Deliverables

See the Lab Manual for information about getting and submitting projects. We will grade only what was submitted before the deadline. If you replaced some of your files with newer versions before the deadline, we will grade only the most recent submission of each unless you fill out the relevant form before the submission deadline.

We will not grade files submitted after the deadline has passed, nor will we grade files submitted via email, in paper form, or in any other manner.

You are responsible for submitting the version of your project that you want graded. We will grade only what you submitted before the deadline. Accidentally submitting the wrong version, or forgetting to submit files, will not be considered grounds for a regrade.

This project is not included in the late work policy for this course. It needs to be completed on schedule and submitted before the due date above. Submissions beyond that deadline will not be considered.

Your grade on this project

There are 4 points possible on this project; they are only available by test cases. We will run test cases with the code you submit; each test case is worth some fraction of the grade. In order to have either part graded, that part’s required test cases must pass; there are no points for these, other than that you are eligible to have other test cases we use graded. Parts one and two are each worth two points.

Test cases that take longer than two minutes to run on the grader’s computer may be deemed incorrect runs, even if a longer amount of time available to them would cause a correct answer. Note that the grader’s computer may be slower than yours, especially if you have a very new computer, such as an M1/M2. Do not assume that the speed on your computer will match that on the grader’s!

We will only choose test cases that can reasonably run in this time; for example, there will be no n=50 cases: no matter how good a programmer you are, we do not expect that you will be able to write code that solves problem in such a manner that the n=50 case produces a correct output within our time constraints. Do not worry: your professor cannot do that either. We will deal with larger cases in later assignments with different techniques.

There is also a style concern for this quarter: under no conditions may your code contain the following declaration: using namespace std; Projects that contain that declaration are worth zero points, regardless of other considerations. This rule is in effect for all projects in CompSci 260P this quarter.

For more information about grading, see the relevant section of the lab manual.