**Algorithms EECS 336** **Problem Set #5**
**Date:** Friday, February 14, 2020 **Due:** Thursday, February 20, 2020

**Submit your solution on Canvas.**
**Do not discuss these problems with other students. You should solve these problems on your own.**

**Problem 1.** A limousine company Limo336 asked you to develop a program that selects the optimal set of trips for each driver. Every morning, each driver receives a list of trip requests for that day. Each request $i$, specifies the time $t_i$ when the passenger $\#i$ should be picked up, the origin of the trip $a_i$ (the location where the passenger should be picked up), the destination of the trip $b_i$ (the location where the passenger should be dropped off), and the payment $p_i$ the driver receives for the trip. The algorithm should find a feasible set of requests $S$ that maximizes the total payment the driver receives by the end of the day:

$$\max \sum_{i \in S} p_i.$$

For simplicity, assume that *there is only one driver.* The set of requests should satisfy the following conditions:

1. Each trip should start exactly on time i.e., trip $i$ (if accepted) should start at time $t_i$.
2. The driver can accept request $i$ only if he/she completes the previous trip before trip $i$ starts.
3. Moreover, the driver should have enough time to get from the destination of the previous trip to the origin of the next trip.

The algorithm has access to the function `travel_time(`$a$`,`$b$`)` that returns the travel time from location $a$ to location $b$. Assume that the travel time only depends on $a$ and $b$, and it is symmetric i.e., `travel_time(`$a$`,`$b$`)` $=$ `travel_time(`$b$`,`$a$`)`. The first trip can start at any location.

**Example.** *The driver receives the following requests:*

1. *10:00am: from the O'Hare International Airport to Northwestern University, payment: $85;*
2. *10:10am: from Northwestern University to the O'Hare International Airport, payment: $90;*
3. *10:50am: from the O'Hare International Airport to UChicago, payment: $120;*
4. *11:10am: from the O'Hare International Airport to Northwestern University, payment: $75;*
5. *12:00pm: from UChicago to Northwestern University, payment: $100.*

*Suppose that it takes 40 minutes to get from the airport to Northwestern; it takes 50 minutes to get from the airport to UChicago; and it takes 55 minutes to get from Northwestern to UChicago.*

*In this example, the driver can accept Request 1. If he does so, he cannot accept Request 2, because he will not complete Trip 1 by 10:10am. He also cannot accept Requests 3 and 4, because even though he will complete his first trip by 10:50am, he will not be able to get back to the airport by 10:50am or 11:10am. However, he may accept Request 5, because he has enough time to get to UChicago by 12:00pm. If the driver accepts Requests 1 and 5, he will earn* $\$185 = \$85 + \$100$*. Note that this is not the optimal solution. The driver can earn more by accepting requests 2, 3, and 5.*

Your goal is to design a dynamic programming algorithm that solves this problem.

1. Describe the subproblem for your dynamic program.
2. Write a recurrence relation for your dynamic program.
3. What is the base case of your program?
4. Explain why your recurrence relation is correct.

**Problem 2.** Consider a binary tree $T$. We say that $T$ is a red-purple tree if the following conditions hold:

1. Each node is either red or purple.
2. Exactly one child node of every purple node $v$ is purple unless $v$ is a leaf.

You are given a weighted binary tree $T$. Your goal is to assign colors – red and purple – to the vertices of $T$ to get a red-purple tree and maximize the total weight of purple nodes.

The node weights are given to you in array $nodeWeights$. The children of vertex $i \in \{0, \ldots, n-1\}$ are vertices $2i+1$ and $2i+2$ (if they exist).

Implement function `MaxRedPurpleTree` that given $nodeWeights$ returns the value of the optimal solution:

- `int MaxRedPurpleTree(const std::vector<int>& nodeWeights)`

**Instructions for the programming assignment.** Download files

- `student_code_5.h` – this file should contain your solution.
- `problem_solver_5.cpp` – this is the main file in the project (don't edit this file!).
- `test_framework.h` – this is a library responsible for reading and writing data files (don't edit this file!)
- `problem_set_5.in` – this file contains test problems for your algorithm (don't edit this file!)

Place all files in a new folder/directory. Write your code in function `MaxRedPurpleTree`. Also, write your name in the function `GetStudentName`. Both functions are located in file `student_code_5.h`. Compile and run your code. To compile your code do the following.

- If you use GNU C++ compiler, type
  `g++ -std=c++11 problem_solver_5.cpp -o problem_solver_5`
- If you use CLang compiler, type
  `clang++ -std=c++11 problem_solver_5.cpp -o problem_solver_5`
- If you use Microsoft Visual C++ compiler, start `Developer Command Prompt` and type
  `cl /EHsc problem_solver_5.cpp`

Your compiler should be compatible with `C++11`. If you work in the Wilkinson Lab, you need to start developer tools first: Type

- `scl enable devtoolset-4 bash`

Once you compile your code, start your program. Type `./problem_solver_5` on Unix or Mac and `problem_solver_5.exe` on Windows. Make sure that the executable is located in the same folder as file `problem_set_5.in`. Your program will generate `solution_5.dat` that contains solutions to the problems from file `problem_set_5.in`. If your code works correctly, you will get the following message:

- Problem set 5. Your algorithm solved all test problems correctly. Congratulations!
- Don't forget to submit your source code and file `solution_5.dat` via Canvas.

If your code makes a mistake, you may get a message like this:

- `Problem set 5.  Mistake in problem #15.  Correct answer:  4.  Your answer:  12.`

Finally, when your code is ready, submit files `student_code_5.h` and `solution_5.dat` via Canvas. Make sure that you are submitting the latest versions.

**Remark:** If you want to debug your code, please, type `./problem_solver_5 15` on Unix or Mac and `problem_solver_5.exe 15` on Windows. This command will call your function only on one problem – the problem #15 and thus let you debug your code on the problem where your program erred. Note that this command will not generate or update `solution_5.dat`. So before submitting your solution, you need to run your program without any command line arguments.