A1: Running an ISP

Overview

You are running an ISP called Time Cast, the next up and coming internet provider. Your business model is to provide broadband internet and actually give your clients the speeds you advertise. You have surveyed a number neighborhoods and compiled lists of prospective clients, including the internet speed they want and the price they will pay. Fortunate for you, they are sick of their current ISP and will switch to your company on a dime, if you make an offer to them.

In each neighborhood, there is a limit on how much bandwidth the infrastructure you can acquire supports, and a cost to maintain service. Your task is to determine if it is viable to move into a neighborhood. If it is, then you should determine which customers to offer service to (if you cannot offer everyone service).

Deliverables

- **Ungraded/Optional –** Submit a PDF to A1 Design assignment on Autolab containing pseudocode for how you plan to solve the problem. Submission will close on Saturday, so aim to get some thoughts in by then.
    - This will assist us in helping you thinking through the problem.
    - This is not required, but highly recommended so that you start early and design your solution before programming (which you should always be doing anyways).
- **A1.hpp** will be submitted to A1 assignment on Autolab.
    - This will be the file containing your C++ code for the project.
- **A1-analysis.pdf** will be submitted to A1 Analysis assignment on Autolab.
    - This will be a PDF with pseudocode of your solution and an analysis of runtime costs.
    - We recommend typing this.
    - **If you submit a .docx or any other file format you will receive no credit. You can easily save a .docx file as a pdf. Any file that is not a pdf will not be graded.**
    - **This will be due 2 days after code submission.**

Due Date

DUE DATE:

A1-design.pdf: 2/18/2017, at 01:59 (this is Saturday, Feb 18 at 1:59AM/ or for those who stay up late, the continuation of Friday night).

A1.hpp: 2/26/2017, at 01:59 (this is Sunday, Feb 26 at 1:59AM, or for those who stay up late, the continuation of Saturday night).

A1-analysis.pdf: 2/28/2017, at 01:59 (this is Tuesday, Feb 28 at 1:59AM, or for those who stay up late, the continuation of Monday night).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****

Last updated: Tuesday, Feb 14, 2017 at 12:03 AM

Start Early!!!!!!!!

Late Policy

The policy for late submissions is as follows:

- Before the deadline: 100% of what you earn (from your best submission).
- One day late: 50-point penalty (lower your earned points by 50, minimum of 0).
- > 1 days late: 0 points.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline. If you are working late, keep in mind that **submissions will close 24 hours after the original deadline** and you will no longer be able to submit your code.

Objectives

In this assignment, you will familiarize with C++ primitive types, raw arrays, and functions. Additionally, you will have to analyze the runtime of your design. You are provided with the signatures for multiple functions. Your task is to complete the functions and solve the appropriate task.

Useful Resources

Review the lecture notes and provided examples to get an idea for using raw arrays.

Instructions

- Download the file `A1.hpp`.
- Within your file `A1.hpp`, there are 3 functions. You will be required to define the functionality of the first function. The remaining two are for bonus credit (specified below). If you don't wish to complete the functionality for a bonus function, simply remove it from the file.

Required: bool neighborhoodIsViable (…)

This function should take in the customer data and return true if you will break even or profit, or return false if there will be a loss. **All input values are positive numbers** (aside from chosenCust data). If you will return true, values should be set in chosenCust representing a set of customers making this true.

Full function signature:

```
bool neighborhoodIsViable(const int& numCust,
                          const int& maxBandwidth,
                          const int& maintenanceCost,
                          const int* const reqBand,
                          const int* const reqPrice,
                          bool* const chosenCust);
```

Arguments:

- **numCust**: number of customers (positive integer)
- **maxBandwidth**: maximum amount of bandwidth you can support (positive number)
    - When selecting customers, the total bandwidth requested by the customers should be less than this maximum.

- **maintenanceCost**: cost to run network in this neighborhood (positive number)
    - You are looking to see if you have customers you can support within your bandwidth limit who will pay more than or equal to your maintenance cost.
- **reqBand**: array of size numCust.
    - reqBand[i] is the bandwidth customer i wants (positive number).
- **reqPrice**: array of size numCust.
    - reqPrice[i] is the price customer i wants (positive number).
- **chosenCust**: array of size numCust.
    - The values selected to be true in chosenCust should give a total amount of bandwidth less than or equal to max bandwidth. The values selected to be true in chosenCust should give a total amount of price that is greater than or equal to maintenance costs (**you do not need to provide the best subset, simply a subset that profits/breaks even**).

Return value:

- **return true** if some subset of customers can allow you to break even or make a profit.
    - **Update the values in chosenCust** to reflect a subset of customers to offer service.
        - For example, if you chose to offer Customer 0 to have service, Customer 1 to have service, and Customer 2 to have no service, then the values would be

```
chosenCust[0] = true
chosenCust[1] = true
chosenCust[2] = false
```

- **return false** if no subset of customers can make a profit.

Bonus (5 points): bool neighborhoodIsViableMaxProfit (…)

This function should work the same as neighborhoodIsViable, except chosenCust should always represent the best value acquired within the bandwidth limit.

Full function signature:

```
bool neighborhoodIsViableMaxProfit(const int& numCust,
                           const int& maxBandwidth,
                           const int& maintenanceCost,
                           const int* const reqBand,
                           const int* const reqPrice,
                           bool* const chosenCust);
```

Arguments:

- **numCust**: number of customers (positive integer)
- **maxBandwidth**: maximum amount of bandwidth you can support (positive number)
    - When selecting customers, the total bandwidth requested by the customers should be less than this maximum.
- **maintenanceCost**: cost to run network in this neighborhood (positive number)

    - You are looking to see if you have customers you can support within your bandwidth limit who will pay more than or equal to your maintenance cost.
- **reqBand**: array of size numCust.
    - reqBand[i] is the bandwidth customer i wants (positive number).
- **reqPrice**: array of size numCust.
    - reqPrice[i] is the price customer i wants (positive number).
- **chosenCust**: array of size numCust.
    - The values selected to be true in chosenCust should give a total amount of bandwidth less than or equal to maxBandwidth. The values selected to be true in chosenCust should give a total amount of price that is greater than or equal to maintenance costs
    - **This should be the set that provides the maximal income (even if it is a loss), that can be made with the bandwidth limit.**

Return value:

- **return true** if some subset of customers can allow you to break even or make a profit.
    - **Update the values in chosenCust** to reflect the maximum income that could be generated within the bandwidth limit.
- **return false** if no subset of customers can make a profit.

Bonus (10 points): bool neighborhoodIsViablePartialBand(…)

This function should work the same as neighborhoodIsViableMaxProfit, but with a big difference. Your clients here are quite desperate and are willing to accept a fraction of the bandwidth requested, as long as you prorate the price. Suppose you have a maximum bandwidth of 2.5 mbps and only one customer in the neighborhood, Customer 0. If Customer 0 asks for 5 mbps for $60 a month, this would put you over your bandwidth limit, but you can offer them 2.5 mpbs of bandwidth, and then you would only charge them $30 per month.

Full function signature:

```
bool neighborhoodIsViablePartialBand(const int& numCust,
                          const int& maxBandwidth,
                          const int& maintenanceCost,
                          const int* const reqBand,
                          const int* const reqPrice,
                          double* const chosenCust);
```

Arguments:

- **numCust**: number of customers (positive integer)
- **maxBandwidth**: maximum amount of bandwidth you can support (positive number)
    - When selecting customers, the total bandwidth requested by the customers should be less than this maximum.
- **maintenanceCost**: cost to run network in this neighborhood (positive number)
    - You are looking to see if you have customers you can support within your bandwidth limit who will pay more than or equal to your maintenance cost.
- **reqBand**: array of size numCust.

- reqBand[i] is the bandwidth customer i wants (positive number).
- **reqPrice**: array of size numCust.
  - reqPrice[i] is the price customer i wants (positive number).
- **chosenCust**: array of size numCust.
  - chosenCust[i] is in range [0-1], and is the fraction of what you will offer to customer i.
    - 0 means you don't offer them anything. 1 means you give what they asked for.
    - A fraction means you will charge them a prorated price and give a fraction of the bandwidth.

Return value:

- **return true** if some subset of customers can allow you to break even or make a profit.
  - Update the values in chosenCust to reflect which customers will receive service.
    - Values for a chosen customer must range from 0 to 1, inclusive.
    - E.g., can offer someone 0.3333 of the service they wanted (they are desperate).
  - For example, if you chose Customer 0 to have full service, Customer 1 to have half service, and Customer 2 to have no service, then the

```
chosenCust values {1,0.5,0} corresponding to:
chosenCust[0] = 1
chosenCust[1] = 0.5
chosenCust[2] = 0
```

- **return fals**e if no set of customers can make a profit.

Testing

To test your code, you will be provided with a main and a simple test. It is up to you that you test your code. Please note that the provided tests here may or may not be tested when you submit your code. You may compute with the following scenarios:

Possible Input:

```
int numCust = 5;
int maxBandwidth = 150;
int maintenanceCost = 100;
int reqBand[] = {30, 40, 150, 20, 60};
int reqPrice[] = {40, 60, 100, 20, 70};
bool chosenCust[5];
```

`neighborhoodIsViable`:

Calling `neighborhoodIsViable` with these parameters should return **true**.

Possible values of **chosenCust** after call:

- {false, false, true, false, false}
  - reqBand[2] = 150 <= maxBandwidth
  - reqPrice[2] = 100 >= maintenanceCost

- {true, true, false, false, false}
  - reqBand[0] + reqBand[1] = 70 <= maxBandwidth
  - reqPrice[0] + reqPrice[1] = 100 >= maintenanceCost
- {true, false, false, false, true}
  - reqBand[0] + reqBand[4] = 90 <= maxBandwidth
  - reqPrice[0] + reqPrice[4] = 110 >= maintenanceCost
- {false, true, false, false, true}
  - reqBand[1] + reqBand[4] = 100 <= maxBandwidth
  - reqPrice[1] + reqPrice[4] = 130 >= maintenanceCost
- {true, true, false, true, false}
  - reqBand[0] + reqBand[1] + reqBand[3] = 90 <= maxBandwidth
  - reqPrice[0] + reqPrice[1] + reqPrice[3] = 120 >= maintenanceCost
- {true, true, false, false, true}
  - reqBand[0] + reqBand[1] + reqBand[4] = 130 <= maxBandwidth
  - reqPrice[0] + reqPrice[1] + reqPrice[4] = 170 >= maintenanceCost
- {false, true, false, true, true}
  - reqBand[1] + reqBand[3] + reqBand[4] = 90 <= maxBandwidth
  - reqPrice[1] + reqPrice[3] + reqPrice[4] = 150 >= maintenanceCost
- {true, true, false, true, true}
  - reqBand[0] + reqBand[1] + reqBand[3] + reqBand[4] = 150 <= maxBandwidth
  - reqPrice[0] + reqPrice[1] + reqPrice[3] + reqPrice[4] = 190 >= maintenanceCost

Bad values for **chosenCust** after call:

- {true, false, true, false, false}
  - Customer2 already asks for the maxBandwidth, so adding anyone else will exceed what you can support. Even though the income exceeds the maintenanceCost, you cannot offer plans if you cannot provide the service.
  - reqBand[0] + reqBand[2] = 180 > maxBandwidth – *cannot have this*.
  - reqPrice[0] + reqPrice[2] = 140 >= maintenanceCost – *this is okay.*
- {true, false, false, true, false}
  - The amount that Customer0 and Customer3 will pay is not sufficient, so you would lose money
  - reqBand[0] + reqBand[3] = 50 <= maxBandwidth – *this is okay*.
  - reqPrice[0] + reqPrice[3] = 60 >= maintenanceCost – *this is bad, losing money.*

neighborhoodIsViableMaxProfit:

Calling `neighborhoodIsViableMaxProfit` with these parameters should return **true**. The values of **chosenCust** should be {true, true, false, true true}. Giving customers 0,1,3,4 service would maximize profit.

neighborhoodIsViablePartialBand:

Calling `neighborhoodIsViablePartialBand` with these parameters should return **true**. The values of **chosenCust** should be {1, 1, 0, 1, 1}. Giving customers 0,1,3,4 full service would maximize profit.

Possible Input:

```
int numCust = 3;
int maxBandwidth = 90;
int maintenanceCost = 70;
int reqBand[] = {50, 50, 80};
int reqPrice[] = {50, 40, 40};
bool chosenCust[3];
```

`neighborhoodIsViable`:

Calling `neighborhoodIsViable` with these parameters should return **false**.

- No subset of customers allows you to not lose money within your maxBandwidth (your maxBandwidth only allows you to support at most 1 customer).

`neighborhoodIsViableMaxProfit`:

Calling `neighborhoodIsViableMaxProfit` with these parameters should return **false**.

- No subset of customers allows you to not lose money within your maxBandwidth (your maxBandwidth only allows you to support at most 1 customer), so you cannot have a maximized profit.

`neighborhoodIsViablePartialBand`:

Calling `neighborhoodIsViablePartialBand` with these parameters should return **true**.

- The best profit would be {1, 0.8, 0}. This means that we give
  - Customer0 a full connection, so 50 for 50,
  - Customer1 a reduced connection (80% of request), so 40 for 32,
  - This gives a total bandwidth of 90 <= maxBandwidth and a total income of 82 >= 70.

Possible Input:

```
int numCust = 2;
int maxBandwidth = 100;
int maintenanceCost = 100;
int reqBand[] = {50, 50};
int reqPrice[] = {50, 40};
bool chosenCust[2];
```

All three cases will return false. No matter who you choose to offer service to, you cannot recover your maintenance costs.

Submission

Code Submission (90 points):

You must submit your A1.hpp file to the A1 assignment on Autolab.

- Your submission must compile and run on Autolab. I suggest that you compile often as you build your code to avoid difficulties debugging syntax and other errors.

Tests will be run on your file and your score will be reported to you when finished. For this assignment there is not limit on the number of submissions you may perform. That said, you should not be using the submission system to test and debug your code.

Bonus (10 points):

If your runtime is faster than our code's runtime (for the required part), you will receive 10 points bonus if your submission correctly passes grading (sorry, immediately returning doesn't count).

Code Analysis Submission (10 points):

You must submit a pdf with pseudocode of your algorithm and runtime analysis of your code. This submission should be uploaded to the A1-analysis assignment. The due date for the analysis will be two days after the assignment is due.

DUE DATE:

A1-design.pdf: 2/18/2017, at 01:59 (this is Saturday, Feb 18 at 1:59AM/ or for those who stay up late, the continuation of Friday night).

A1.hpp: 2/26/2017, at 01:59 (this is Sunday, Feb 26 at 1:59AM, or for those who stay up late, the continuation of Saturday night).

A1-analysis.pdf: 2/28/2017, at 01:59 (this is Tuesday, Feb 28 at 1:59AM, or for those who stay up late, the continuation of Monday night).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****

AI Policy Overview

As a refresher from what we stated in the first day of lecture, please re-read the academic integrity policy of the course. I will continue to remind you throughout the semester and hope to avoid and incidence.

What constitutes a violation of academic integrity? These first bullets should be obvious:

- Turning in your friend's code (obvious)

- Turning in solutions you found on google (should be obvious)
- Paying someone to do your work (just don't)
- Posting to forums asking someone to solve the problem (Yahoo answers will not save you)

You should know that seeking solutions to the assignment does not fall under solving the problem yourself. Things that may not be as obvious:

- Sending your code to a friend to help them (**if they use your code, you are also liable**)
- Reading your friend's code (it will most likely influence you directly or subconsciously to solve the problem identically)

The assignments should be solved individually with assistance from course staff. You may discuss and help one another with technical issues, such as how to get your compiler running, etc. It is not acceptable that you both worked together and have nearly identical code. If that is going to be a problem for you, don't solve the problems in that close of proximity.

There is also a gray area. How far can you discuss the problems. **I do encourage you to work with one another to solve problems**. That is the best way to learn and overcome obstacles, but at the same time you need to be sure you do not overstep. Talking out the solution from a high level is okay (I choose to use stack to store ___ and then went through things in thus and such order). Just remember that you shouldn't be showing code for **how** to do that. A good rule of thumb would be: Don't come away from discussions with your peers with any written work. Especially not written code.

If you have concerns that you may have overstepped or worked closely with someone, please address me prior to deadlines for the assignment. We will address options at that point.

All this said, please feel free to use any files that we provide directly in your code. Feel free to directly use anything from lecture or recitations. You will never be penalized for doing so. If you are in doubt, always cite where your code comes from. Just remember, if you are citing an algorithm that is not provided by us, then you are probably overstepping.