# CS401: Dynamic Programming Project

Due: Tuesday, Oct 27 at 3:29PM.
Teams of two are allowed.

---

## The Problem: Synthesis of Trees of Gates

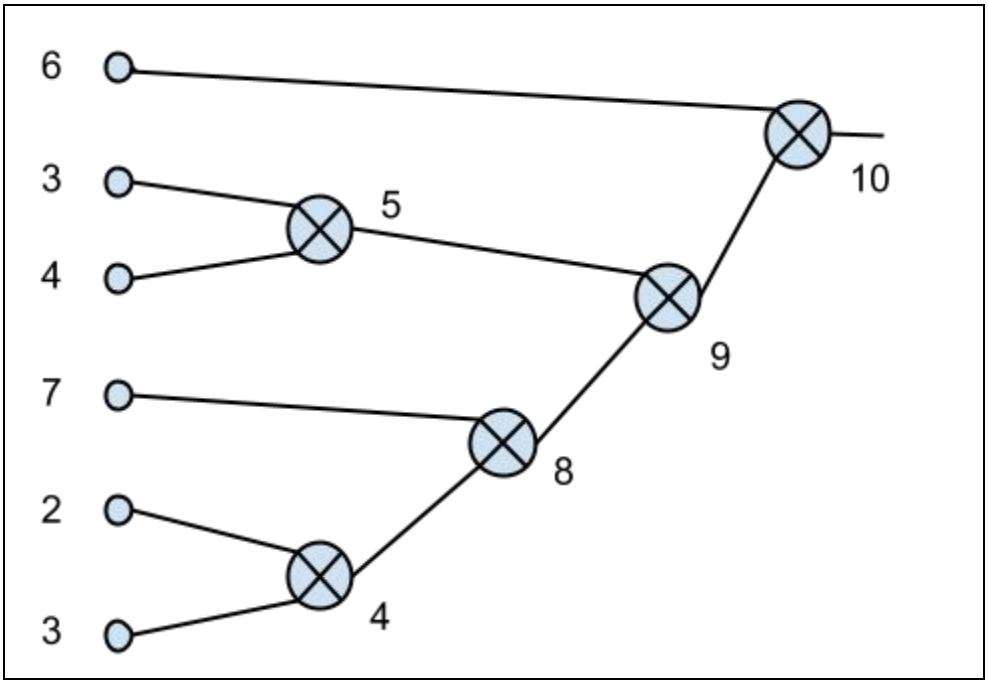
Suppose there are $n$ input signals/bits $b_0, b_1, ..., b_{n-1}$ and you need to build a tree of 2-input gates to compute the XOR of the bits (or, for the purposes of this problem, an AND, OR, or any other associative operator).

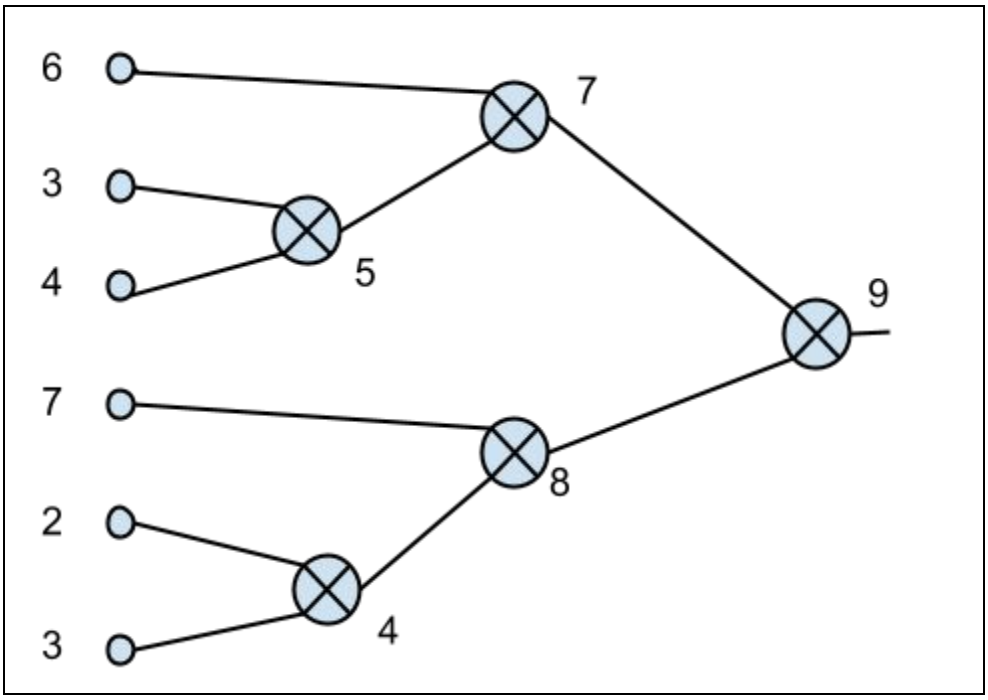
Signal $i$ arrives at its input pin at time $t_i$. The arrival time at the output of a gate $g$ with inputs $x$ and $y$ is determined by:

$$t_g = \max(t_x, t_y) + d_g$$

where $t_x$ and $t_y$ are the arrival times of inputs $x$ and $y$ respectively and $d_g$ is the delay through the gate.

The figures below give two trees for the given input. The input pins are at the left of the diagrams and are labeled with their arrival times. The two configurations






**Allowable Trees**

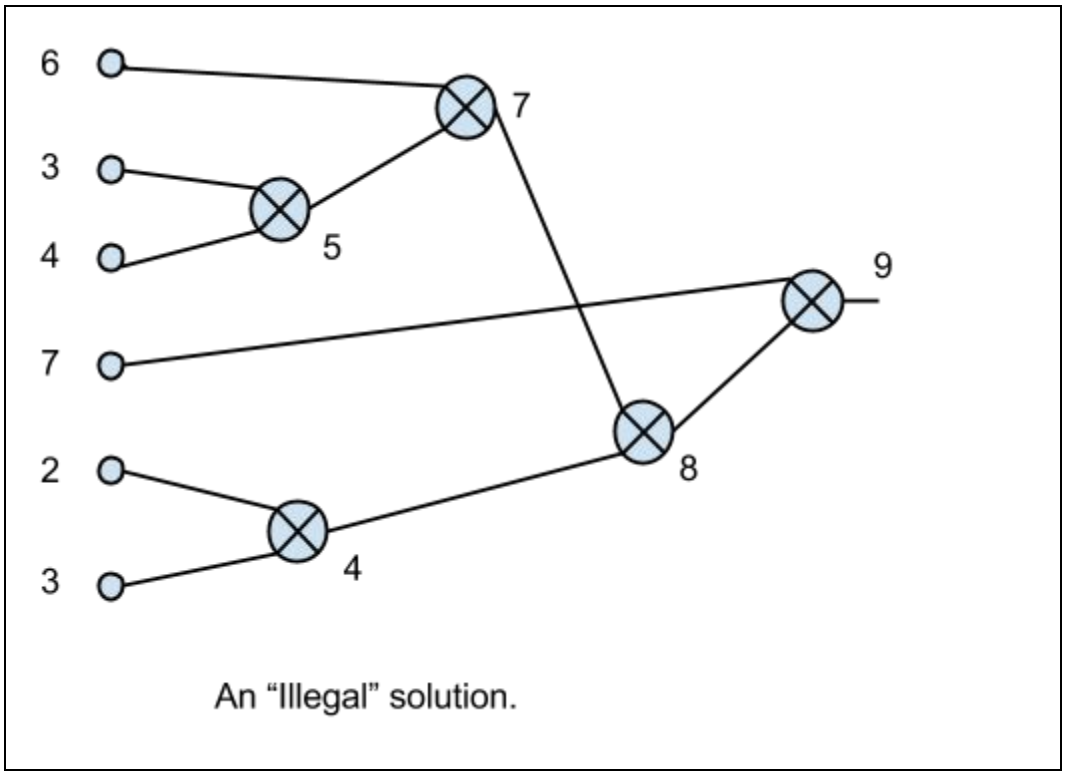
Because of the given ordering of the inputs, we will not allow just any tree structure; the trees must be “consistent” with the input pin ordering:

> The leaves (inputs) of every subtree must be a consecutive subsequence of length two or more from the given pin ordering.

The tree in the diagram below violates this rule.

Don’t worry, this is good news for you:  it makes the problem more tractable.

---



An "Illegal" solution.

## Problem Formulations

There are two formulations you may consider for differing levels of credit.

### Formulation 1 (Up to 80% credit for teams of size 1; 50% for teams of size 2):

**Given:** A sequence of $n$ arrival times $t_0, t_1, ..., t_{n-1}$, and a gate delay $d_g$ (as a float).

**Objective:** Construct a legal tree of gates which minimizes the arrival time at the output of the tree.

In other words, make it as fast as possible regardless of cost.

### Formulation 2 (Up to 120% credit for teams of size 1; 100% for teams of size 2):

In this more realistic formulation, we have a library of gates which have different tradeoffs between gate delay and cost (e.g., area, power consumption…).

**Given:**

- A sequence of $n$ arrival times $t_0, t_1, ..., t_{n-1}$,
- a library of $m$ logically equivalent gates $\{(d_0, c_0), (d_1, c_1), ..., (d_{m-1}, c_{m-1})\}$

- a maximum allowable arrival time at the output of the tree $t_{max}$

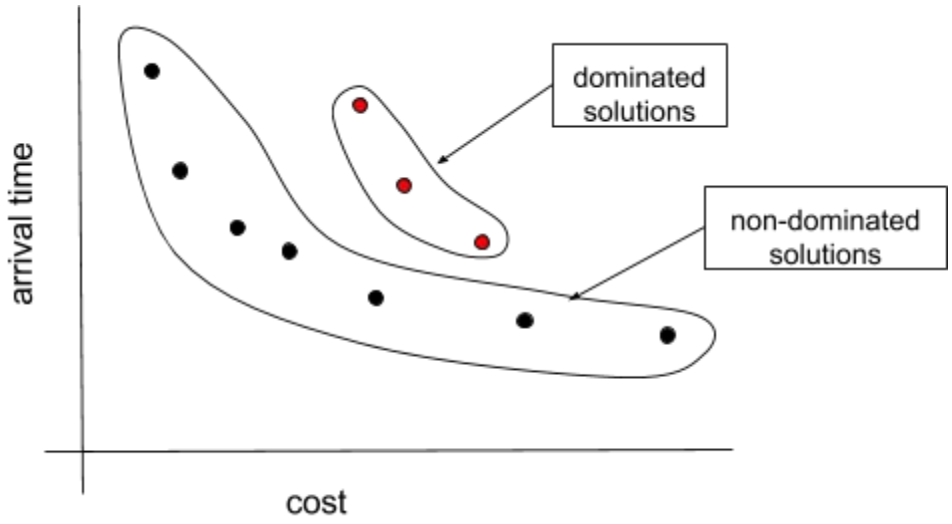
**Objective:** Construct a legal tree of gates which minimizes the total cost of the tree subject to the constraint that the arrival time at the output is no greater than $t_{max}$.

## Formulation 3 (Up to 140% credit for teams of size 1; 120% for teams of size 2):

This is a generalization of Formulation. It also accounts for tradeoffs between arrival time and cost, but does ***not*** take a maximum arrival time constraint as input.

Instead, it generates a full tradeoff curve of "non-dominated" candidate solutions from which a user can select. A solution is non-dominated if there is no other solution that has ***both*** lower cost and smaller arrival time. The set of all such solutions forms a tradeoff curve (sometimes called a pareto curve).

The concept is illustrated in the diagram below. The configurations marked in red are intrinsically sub-optimal and so need not be considered at all. The configurations marked black are non-dominated and together capture the intrinsic cost-vs-arrival time tradeoff of the problem instance.




For completeness:

**Given:**

- A sequence of $n$ arrival times $t_0, t_1, ..., t_{n-1}$,
- a library of $m$ logically equivalent gates $\{(d_0, c_0), (d_1, c_1), ..., (d_{m-1}, c_{m-1})\}$

**Output:** produces the set of non-dominated candidate solutions as a tradeoff-curve.

## Formats

**Pin Arrival Times and Ordering:** The first file format specifies the signal arrival times in sequence. The inputs are implicitly labeled $0..n-1$. The file is as simple as can be:

- the first line specifies the number of input pins $n$;
- this is followed by $n$ lines containing $t_0, ..., t_{n-1}$ as floating point numbers.

The problem instance used in the preceding diagrams would look like this:

```
6
6.0
3.0
4.0
7.0
2.0
3.0
```

**Gate Library:** A gate library is specified in a file organized as follows:

- The first line is the integer $m$ -- the number of gates in the library.
- This is followed by $m$ lines, describing gates $0..m-1$. each line contains two positive numbers (in this order).
    - the delay -- a floating point number
    - the cost -- an integer
- The ordering of the gates implicitly determines their IDs -- $0..m-1$
- The gates are ordered in *decreasing* order of delay and *increasing* order of cost: slowest/cheapest gate first.

For example, a library with three gates might look like this:

```
3
5.0 1
4.5 1
3.5 3
```

**Topology Format: postfix notation**

A solution is a binary tree in which the leaves are labeled

```
p0 p1 ...
```

Each internal node represents a gate and is labeled:

```
gX
```

where X is the gate ID from the library (or simply 0 if there is only one such gate).

The structure of the tree itself is specified in postfix notation on a single line. Think of the gates as operators and as the inputs as operands.

The topology from the first figure would be specified as:

```
p0 p1 p2 g0 p3 p4 p5 g0 g0 g0 g0
```

Your program will use this format when reporting a configuration.

## Usage

Your program should run from the command line as follows:

```
% gtree <pin-file> {<library-file>} {-t <tmax>}
```

The pin file is required.
If no library file is specified, the default library is just a single gate with unit delay and unit cost.
If no maximum arrival time `tmax` is specified, the program should minimize the arrival time.

For example the following would run the program on a pin file test1 using gate library lib1 with timing constraint 10.0

```
% gtree test1 lib1 -t 10.0
```

**Formulation 1 output:**

- The arrival time of the solution found
- The cost
- The topology in postfix notation

**Formulation 2 output:**

- Whether a feasible solution exists or not
- If a solution exists:
    - The arrival time of the solution
    - The cost
    - The topology in postfix notation

**Formulation 3 output:**

- All non-dominated solutions in increasing order of cost (and decreasing order of arrival time).
- Each solution will occupy two lines of output:
    - A line stating the cost and arrival time of the solution followed by
    - A line giving the corresponding topology in postfix notation.

---

**Assumptions**: For this assignment, you may assume the following:

- Input files are well-formatted
- The command line arguments are in *exactly* the order specified above.

## Language

The project may be completed in the language of your choice provided your program can be run as specified above. I'm expecting most of you will use C, C++ or Java, but other choices should be ok. Some might choose Python which I guess is ok, but probably not the most realistic language to implement optimization algorithms that may become compute intensive.

## Deliverables

Each team will submit two items for this project:

- A report containing the following:
    - **Name(s)** of team member(s)
    - **Summary** of accomplishments:
        i. Did you complete a working program?
        ii. Which formulation did you solve?
        iii. Are there any bugs or uncertainties you have about your implementation? Any limitations?
    - **Division of Labor**: what did each team member contribute?

    - **Algorithm Description:** of your algorithm including argument of correctness and a runtime analysis. You should not cut and paste code from your implementation here!
    - **Solution Extraction:** A description of how you construct the the optimal topology.
    - **Details**: include any significant implementation details -- e.g., did you use memoization, etc.
    - **Example Instance:** An example of an interesting problem instance on which you ran your code. Include the input files and the output report.
- Your source code including a **`readme`** file containing compilation instructions. A `Makefile` would be great! Also include the input files used in your report example.

Your report will be submitted as a hardcopy in class. All other material will be submitted electronically.

## Notes

The gate costs are specified as *integer* rather than floats. Your algorithm should work correctly if we were to change them to floats. However, assuming they are given as integers does allow you to bound the number of distinct costs that are possible which, in turn, may play a role in your runtime analysis.