CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 8 Feb 1, 2013

Computational Complexity

Upcoming Homework Assignment

HW #2 **Due: Tonight 6pm**

Binary Search Trees

Remember, the name of the game here is *recursion*. If you have long expansive functions, they may technically give the right output, but they involve more code than is necessary. Recursive functions are (usually) short and elegant. You'll need to know this stuff for tests, for future homeworks, and for your own sanity in later courses. If you've worked a problem iteratively, that's cool, but make sure you can do it recursively, too.

Lecture Goals

1. Announcements
2. Computational Complexity
3. Homework 3

Announcements

The errors are:

g++ -Wall -o "binary_search_tree" "binary_search_tree.cpp" (in directory: /home/user/Documents/Homework 2)
/usr/bin/ld: /usr/lib/debug/usr/lib/i386-linux-gnu/crt1.o(.debug_info): relocation 0 has invalid symbol index 11
/usr/bin/ld: /usr/lib/debug/usr/lib/i386-linux-gnu/crt1.o(.debug_info): relocation 1 has invalid symbol index 12
/usr/bin/ld: /usr/lib/debug/usr/lib/i386-linux-gnu/crt1.o(.debug_info): relocation 2 has invalid symbol index 2
/usr/bin/ld: /usr/lib/debug/usr/lib/i386-linux-gnu/crt1.o(.debug_info): relocation 3 has invalid symbol index 2
/usr/bin/ld: /usr/lib/debug/usr/lib/i386-linux-gnu/crt1.o(.debug_info): relocation 4 has invalid symbol index 11
(and so on)
/usr/lib/gcc/i686-linux-gnu/4.6/../../../i386-linux-gnu/crt1.o: In function `_start':
(.text+0x18): undefined reference to `main'
/tmp/ccfRvWhM.o: In function `insert_data(bt_node**, int)':
binary_search_tree.cpp:(.text+0xd2): undefined reference to `insert(bt_node**, bt_node*)'
collect2: ld returned 1 exit status
Compilation failed.

This means you have stale object files. ‘make clean’ and then ‘make’ to refresh everything.

Announcements

```
$ python basic_functions.py
  File "basic_functions.py", line 31
    return 4
    ^
IndentationError: unexpected indent
```

This means you (or your editor) have inserted spaces when the interpreter expected tabs. Only defense against this is to be really careful, and to compile/interpret frequently.

Announcements

By the way, you can compile a Python script to check for syntax errors *without running the program*:

```
python -m py_compile basic_functions.py
```

#up_to_date

There needs to be a quick way of getting info out about non-critical info about the class, or about Retrograde, or little helpful hints about homework.

Follow **@cs1300_cs2270** for updates.

I'll still send email if something major happens, so this isn't required.

Computational Complexity

This is a mathematical way of reasoning about how much *time* (or *operations)* or *space (memory)* an algorithm or data structure requires.

Thought Experiment

Say I have a gigantic list of numbers. Millions of them. Seventeen digit beasts. They are not really in any sort of sensible order.

How many numbers do you expect to look at before finding the number 37438432029 34?

Or concluding that it isn't in there?

Big Oh

With an unsorted list of numbers, you might have to look through all of them to find it. You'll certainly have to look through all of them to determine it isn't there, and that I've been wasting your time.

In other words:

Given an unsorted array of size **n**, the algorithm to find a specific value is written as **O(n)** and pronounced 'big oh of n', or just 'oh of n'.

What's the Complexity?

Say you are confronted with the following code:

```
for (int i=0; i < N; i++) {
  for (int j=0; j < N; j++) {
    product[i][j] = i * j;
  }
}
```

What's the computational complexity of this algorithm? How many times will we run the inner loop?

Survey Says...

```
for (int i=0; i < N; i++) {
 for (int j=0; j < N; j++) {
  product[i][j] = i * j;
 }
}
```

The runtime complexity of the above is **$O(n^2)$**. Stare at this and convince yourself I am not lying. (I'm not, this time.)

Complexity of sorted things

Say we have a *sorted* data structure containing a billion numbers.

How many slots *on average* do we have to look in to determine if it has a particular value?

How many slots *in the worst case scenario* do we have to look in? The *best case*?

Complexity of BSTs

If the sorted data structure is a plain binary search tree, the best/average/worst-case scenarios are:

Best: **O(1).** The droid we are looking for is on top.

Average: **O(log n).** Happens when the tree is balanced.

Worst: **Survey the audience?**

Complexity of BSTs

(answer from previous: **O(n)**)

How about the insert function?

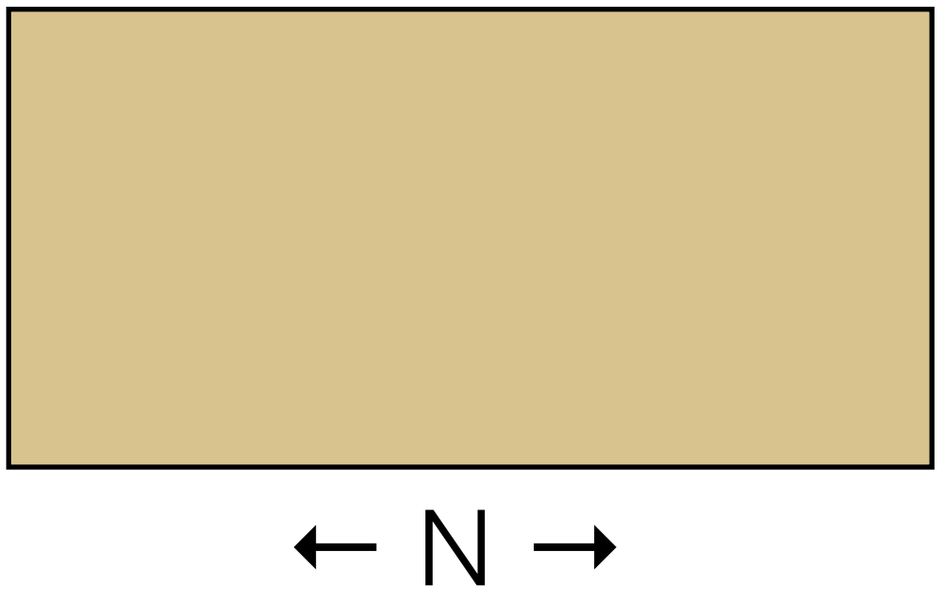
Remove?

Size?

Consider this an ungraded homework assignment. Wikipedia is *wonderful* in this regard.

Geometry Can Help

```
for (int i=0; i < M; i++) {
 for (int j=0; j < N; j++) {
  product[i][j] = i * j;
 }
}
```

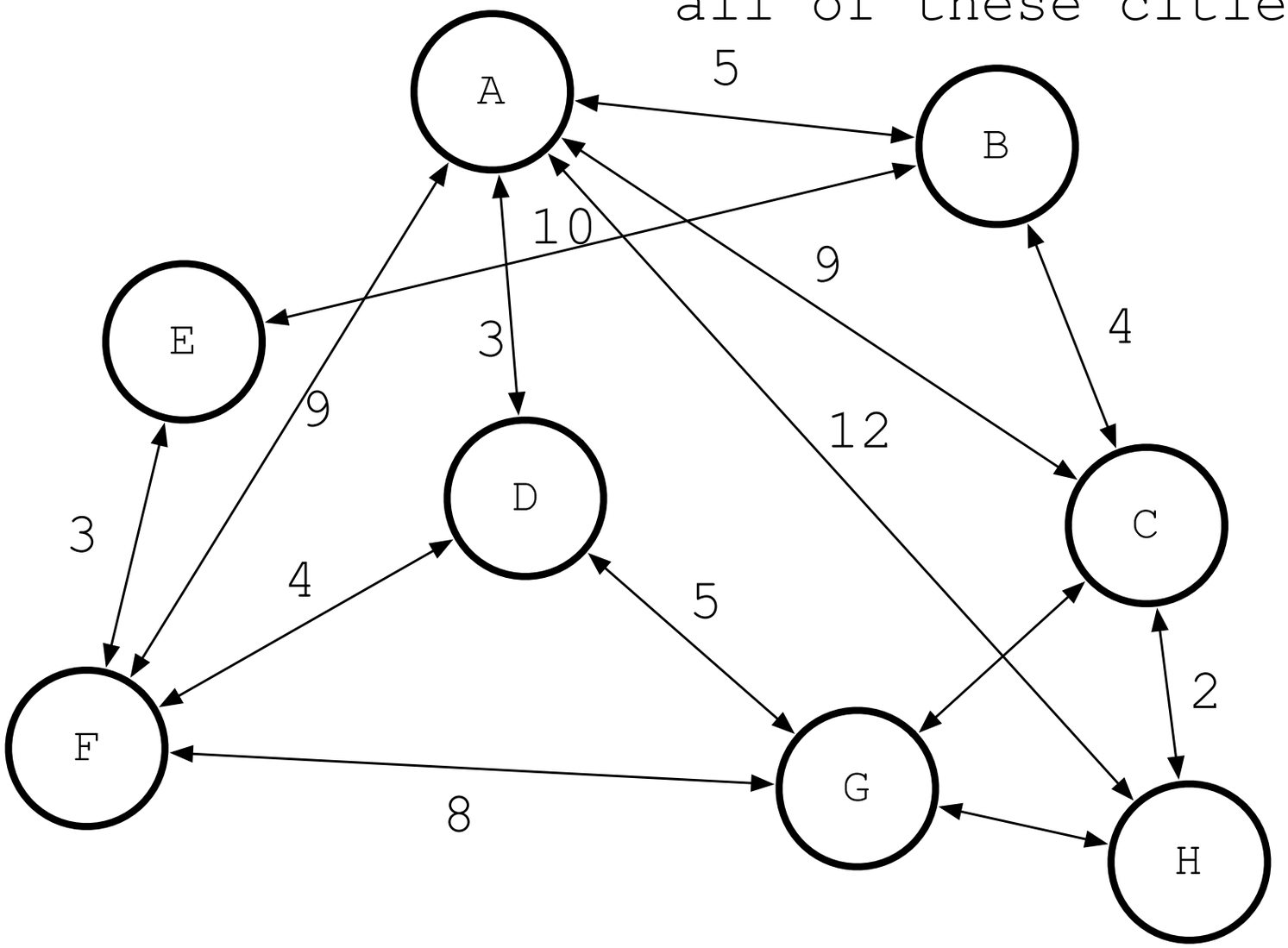


This is O(M*N). It is not common to have two terms, but it can be done. Often if you know what M and N mean, that can point you to where you need to write really efficient code.

Really Hard Problems

Traveling Salesman Problem
aka **TSP**:

Starting from A, what's the 'best' path to tour all of these cities?



O(n!)

This is an example of an NP-Hard problem, and has driven CS graduates students insane since the inception of computer science.

Really Hard Problems II

Crytography is one area where math and CS people actively look for algorithms that go beyond merely being tricky, but require more computing power than is available to solve.

E.g. if you don't have the secret key (or a hint) to decode an encrypted message, a brute force attack might be **$O(2^n)$** where **n** is the number of bits in the decryption key. This means if you want to make it twice as hard to crack, add a single bit.