CS 361
Data Structures & Algs
Lecture 6

Prof. Tom Hayes
University of New Mexico
09-06-2012

Reminders

- Written HW #1 is now past due.
- Late HW: 10% deduction per day late, except for valid emergencies.
- Written HW #2, due Thursday 9/20:
  - problems 1.7, 1.8, 2.1, 2.2, 2.3, 2.4
- Programming: Implement a Stable Matcher. Due Thursday 9/27.
- Reading: Finish Chapter 2 this weekend.

Programming #1

- Should be able to read preference lists from an input file specified on command line.
- Each line of the input file will be (n+1) names separated by whitespace. For instance, Alan Betty Carol Dora means Alan ranks Betty first and Dora last.
- Output: any stable perfect matching.

Last Time

A Yahtzee!-like problem (all red/black)

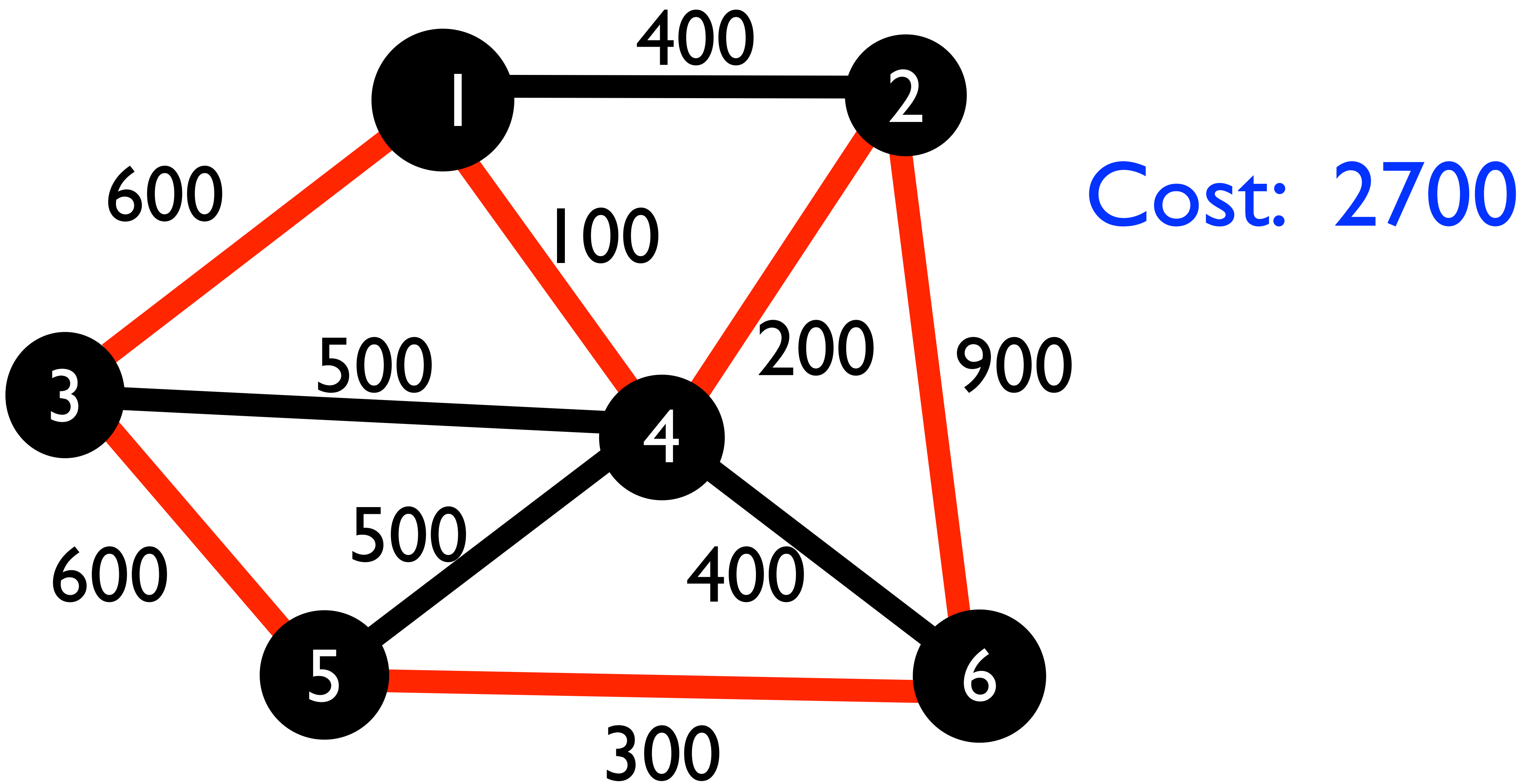
Traveling Salesman

- Brute force: (n!) possible tours
- DP: ($n^2 2^n$) subproblems
- NP (Nice Proofs, may be hard to find)

Running times.

“Big O” notation.

Traveling Salesman



Find: Loop visiting each town exactly once.

Minimize total cost.

TSP Recap

Brute force: n! possible solutions.

$$n! \approx 2^{n \log n}$$

Better: Divide into two “halves” of size roughly n/2. Find the shortest route through each half, recursively using the same idea.

Dynamic programming: keep track of solutions to all sub-problems, and re-use when possible. (i.e., memoize)

Analysis, DP solution

How many sub-problems will we look at, total? (i.e., throughout the entire recursion)

Each looks like this: start city, intermediate cities, end city.

$\leq n^2 \, 2^n$ possibilities

Time to solve one sub-problem? Loop through all the ways to split it into 2 subproblems. Sum up the values for those, keeping the min. constant * $n^2 \, 2^n$

TSP, final thoughts

Improved from (n!) brute force, to roughly $n^2\ 4^n$ via our dynamic programming alg. How big an improvement?

$$\log(n!) \approx n \log n$$

$$\log(n^2 4^n) \approx 2n$$

NP

NP is the class of YES/NO decision problems, where, for every input of size n, if the correct output is YES, then there exists an efficiently checkable proof of that fact.

TSP asks, “Is there a tour of these cities that costs at most M?”

This is in NP, because, when the answer is YES, then, if I give you the tour, you can verify the YES answer. This does not mean finding such a tour can be done quickly.

NP

NP is the class of YES/NO decision problems, where, for every input of size n, if the correct output is YES, then there exists an efficiently checkable proof of that fact.

We say a problem is *NP-hard* if it could be used to solve every problem in NP.

Say a problem in *NP-complete* if it is NP-hard *and* a member of NP.

P vs NP

This multimillion-dollar problem asks, are there any problems in NP, for which it is not possible to efficiently determine the answer when a proof is NOT GIVEN?

In other words, is it easier to verify a proof than to come up with one on your own?

TSP is *NP-complete*. This means, if you can find an efficient algorithm to solve it, you have proven every problem in NP is easy.

Efficient Algorithms

Our Dynamic Programming algorithm for TSP was a big improvement, but is still not efficient.

To be efficient, when the input size is n, our algorithm should run in, say, time 10n, or perhaps $50n^2$, or $100n^3 + 50n \log(n)$. To be general, let's say any function that is less than some power of n, such as n^{10}. For short, poly(n).

Big O notation

We want to be able to reason carefully about running times, but without "sweating irrelevant details."

Who cares if the running time is n^3 versus $n^3 + 10.5\, n^2 - 0.5n$? It can matter only for a few small values of n. In the "big picture" what really matters is, approximately how big an input can I handle in the trillion or so steps I have time to do.

Big O helps us achieve these goals.

Big O, formally

Suppose g is a function describing a running time. g(n) tells us the amount of time our program runs on an input of length n. The notation O(g) refers to the class of all functions that, for large inputs, do not grow faster than a constant times g. In other words, a function f is in O(g) if there exists a constant C such that, for every n, f (n) $\leq$ C g(n). *see caveat in a few slides.

Big O, informally

One generally writes “f = O(g)” to indicate that f is in the function class O(g). This is just a shorthand, and can lead you into trouble if you try to use any laws of “=”.

For instance, it would be correct to write $20n^2 + 6\ n = O(n^3)$ and also to write $20n^2 + 6\ n = O(n^2)$. However, $O(n^2)$ and $O(n^3)$ are not equal. Exercises 2.5(ab) illustrate some further pitfalls. Also, see wikipedia on Big-O (not the anime!)

Caveat - zeros

$10(n-1)^3 = O(n^4)$. Why?

But, is $10n^3 = O((n-1)^4)$?

We want it to be.

But, for n=1, there is no constant C that could work. Why? $(1-1)^4 = 0$.

Fancier definition: $f = O(g)$ means there exists C, n_0, such that, for every $n \geq n_0$, $f(n) \leq Cg(n)$.

Why define it like that?

f = O(g) means:

There exists $C>0$ and n_0 such that, whenever $n \geq n_0$, we have $f(n) \leq C\, g(n)$.

(1) Simplifies analysis: A sequence of O(1) “atomic” steps (no recursive function calls or loops) can be replaced by a single “step” conceptually.

(2) Gets at big question: limiting growth rates for f and g.

More on Big-O

f = O(g) is a 1-sided guarantee!

We know “f is not much bigger than g (for large inputs)” but we don’t know whether “g is much bigger than f (for large inputs)”

This is a good thing! (Less to prove)

Q: What if we want a 2-sided guarantee?

A: Ω, Θ notation

Ω, Θ notation

Ω: Omega Θ: Theta (Greek, upper case)

f = Ω(g) means g = O(f). That is, g is (up to a constant factor, and for large inputs) a lower bound on f.

f = Θ(g) means both f = O(g) and f = Ω(g). That is, f and g “are of the same order”

Practice with big-O

How to prove that $5 n + 2 = O(n)$?

Reasoning: $5 n + 2 <= C n$ (goal)

Try $C = 6$. Plug in: $5 n + 2 \leq 6 n$ solve

$2 \leq (6 - 5) n = n$. Choose $n_0 = 2$.

We're now ready to fill in proof.

Practice with big-O

How to prove that $5n + 2 = O(n)$?

Proof: Let $C = 6$. Let $n_0 = 2$.

Assume $n >= n_0$. Then

$5n + 2$

$= 5n + n_0$ (def of n_0)

$<= 6n$ (since $n \geq n_0$)

$= Cn$. (def of C)

Therefore, $5n + 2 = O(n)$ by definition.

Practice with big-O

How to prove that $\log(3n^2) = O(\log(n))$?

Reasoning: $\log(3n^2) \leq C \log(n)$ (goal)

LHS = $\log(3) + \log(n^2) = \log(3) + 2 \log(n)$

Goal: $\log(3) + 2 \log(n) \leq C \log(n)$.

Take $C = 3 > 2$. Solve

$\log(3) + 2 \log(n) \leq 3 \log(n)$ for n, to find n_0

$\log(3) \leq \log(n)$. So $n \geq 3$. Take $n_0 = 3$.

Practice with big-O

How to prove that $\log(3n^2) = O(\log(n))$?

Proof: Choose $C = 3$ and $n_0 = 3$.

Suppose $n \geq n_0$.

$\log(3n^2)$

$= \log(3) + 2 \log(n)$ (arithmetic)

$= \log(n_0) + 2 \log(n)$ (def of n_0)

$\leq \log(n) + 2 \log(n) = 3 \log(n)$ (since $n \geq n_0$)

$= C \log(n)$ (def of C). Thus $f = O(g)$

Practice with big-O

Suppose $f = O(g)$ and $g = O(H)$.

Prove: $f = O(H)$.

Reasoning: Goal: $f(n) \leq C\, H(n)$.

$f = O(g)$ means: There is C_1, n_1 such that as long as $n \geq n_1$ we have $f(n) \leq C_1\, g(n)$.

$g = O(H)$ means: There is C_2, n_2 such that as long as $n \geq n_2$ we have $g(n) \leq C_2\, H(n)$.

$f(n) \leq C_1\, g(n) \leq C_1\, (C_2\, H(n)) = (C_1\, C_2)\, H(n)$

Practice with big-O

$f = O(g)$ means: There is C_1, n_1 such that as long as $n \geq n_1$ we have $f(n) \leq C_1\, g(n)$.

$g = O(H)$ means: There is C_2, n_2 such that as long as $n \geq n_2$ we have $g(n) \leq C_2\, H(n)$.

$f(n) \leq C_1\, g(n) \leq C_1\, (C_2\, H(n)) = (C_1\, C_2)\, H(n)$

Guess $C = C_1\, C_2$

$n_0 = ?$. Need: $f(n) \leq C_1\, g(n)$. From top, need $n \geq n_1$. Need: $g(n) \leq C_2\, H(n)$. Thus need $n \geq n_2$. Choose $n_0 = \max\{n_1, n_2\}$.

Practice with big-O

Suppose f = O(g) and g = O(h).

Prove: f = O(h).

Proof: f = O(g) means: There is C_1, n_1 such that as long as $n \geq n_1$ we have $f(n) \leq C_1\ g(n)$.

g = O(H) means: There is C_2, n_2 such that as long as $n \geq n_2$ we have $g(n) \leq C_2\ H(n)$.

Choose $C = C_1\ C_2$, and $n_0 = \max\{n_1, n_2\}$.

Then

Proof: $f = O(g)$ means: There is C_1, n_1 such that as long as $n \geq n_1$ we have $f(n) \leq C_1 \, g(n)$.

$g = O(H)$ means: There is C_2, n_2 such that as long as $n \geq n_2$ we have $g(n) \leq C_2 \, H(n)$.

Choose $C = C_1 \, C_2$, and $n_0 = \max\{n_1, n_2\}$.

Suppose $n \geq n_0$

Then

$f(n) \leq C_1 \, g(n) \leq C_1 \, (C_2 \, H(n)) = (C_1 \, C_2) \, H(n)$

$= C \, H(n)$.

Thus $f = O(H)$.

Choose $C = C_1 C_2$, and $n_0 = \max\{n_1, n_2\}$.

Suppose $n \geq n_0$

Then

$f(n) \leq C_1 g(n)$ (since $n \geq n_0 \geq n_1$ and above)

$\leq C_1 (C_2 H(n))$ (since $n \geq n_0 \geq n_2$ and above)

$= (C_1 C_2) H(n)$ (arithmetic)

$= C H(n)$. (def of C)

Thus $f = O(H)$.

Choose $C = C_1 C_2$, and $n_0 = \max\{n_1, n_2\}$.

Suppose $n \geq n_0$

Then

$f(n) \leq C_1 g(n)$ (since $n \geq n_0 \geq n_1$ and above)

$\leq C_1 (C_2 H(n))$ (since $n \geq n_0 \geq n_2$ and above)

$= (C_1 C_2) H(n)$ (arithmetic)

$= C H(n)$. (def of C)

Thus $f = O(H)$.

Test your understanding

True or False:

When f, g are positive functions, “$f = O(g)$” means there is some constant C such that, for all n, $f(n)/g(n) \leq C$.

Test your understanding

True or False:

When f, g are positive functions, “f = O(g)” means there is some constant C such that, for all n, $f(n)/g(n) \le C$.

True!

Same as $f(n) \le C\, g(n)$.

But, what about n_0?

Test your understanding

True or False:

When f, g are positive functions, "f = O(g)" means

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = C$$

Test your understanding

True or False:

When f, g are positive functions, “f = O(g)” means

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = C$$

False. f(n)/g(n) does not have to converge to a particular value. C is only an upper bound. See board.

Test your understanding

True or False:

When f, g are positive functions, "f = ϴ(g)" means

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$$

Test your understanding

True or False:

When f, g are positive functions, "f = Θ(g)" means

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = C$$

False. f(n)/g(n) does not have to converge to a particular value. For instance, f(n)/g(n) may oscillate between a lower bound, L, and an upper bound U.

Test your understanding

True or False:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \quad \text{implies } f = O(g)$$

True. Existence of this limit implies that, for large n, f(n)/g(n) is arbitrarily close to C. In particular, f(n)/g(n) is between 0 and 2C. But this implies f(n) ≤ 2C g(n), so f = O(g).

Test your understanding

True or False:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = C \quad \text{implies f = O(g)}$$

True. Existence of this limit implies that, for large n, f(n)/g(n) is arbitrarily close to C. In particular, f(n)/g(n) is between 0 and 2C. But this implies f(n) ≤ 2C g(n), so f = O(g).

Same proof shows f = Ω(g). Hence f=Θ(g)

Setting up a proof

Given: $f = O(g)$. f, g are positive.

Prove: $f^2 = O(g^2)$

Proof:

Setting up a proof

Given: $f = O(g)$. f, g are positive.

Prove: $f^2 = O(g^2)$

Proof: From the hypothesis, there exists C such that, for every n, $f(n) \leq C\, g(n)$.

...

Therefore, for every n, $f^2(n) \leq C'\, g^2(n)$, where C' = ... Thus $f^2 = O(g^2)$.

Setting up a proof

Given: $f = O(g)$. f, g are positive.

Prove: $f^2 = O(g^2)$

Proof: From the hypothesis, there exists C such that, for every n, $f(n) \leq C\, g(n)$.

Square both sides. $f^2(n) \leq C^2\, g^2(n)$.

Therefore, for every n, $f^2(n) \leq C'\, g^2(n)$, where $C' = C^2$. Thus $f^2 = O(g^2)$.