CS152: Introduction to Theory of Computation

- *Instructor*: Eli Upfal (eli)

- *HTA*: Jason Williams (jdw)
- *TA*: Thomas Duffy (tbd)

Mechanics

Texts:

- Sipser, Introduction to the theory of Computation
- ??

Requirements

- required readings
- problem sets *60 –70% of grade*
  - proofs concise and correct
  - typeset in LaTeX
  - late loses 20% per working day.
- 1–2 closed-book exams *30-40%*

What are the fundamental capabilities and limitations of computers?

Three major topics:

- What is a computer? - *Automata Theory*
- What can (cannot) be computed? - *Computability Theory*
- What can (cannot) be computed efficiently? - *Complexity Theory*

Why Study Theory?

pragmatic reasons

- apply efficient algorithms to tractable problems
- avoid intractable or impossible problems
- learn to tell the difference

mathematical education

- automata theory
- complexity classes
- decidability theory

philosophy

- what is computation?
- what is provable?

Topics

- *Automata Theory:* What is a computer? (Review)
- *Computability Theory:* What can computers do? (Semi-review)
- *Complexity Theory:* What makes some problems computationally hard and others easy?
- *Evading intractability*:
  - Approximation
  - Randomization
  - Parallelism
  - ...
- *Lower bounds:* Circuit complexity.

Complexity Theory

Key notion: *tractable* vs. *intractable* problems.

A *problem* is a general question:

- description of parameters
- description of solution

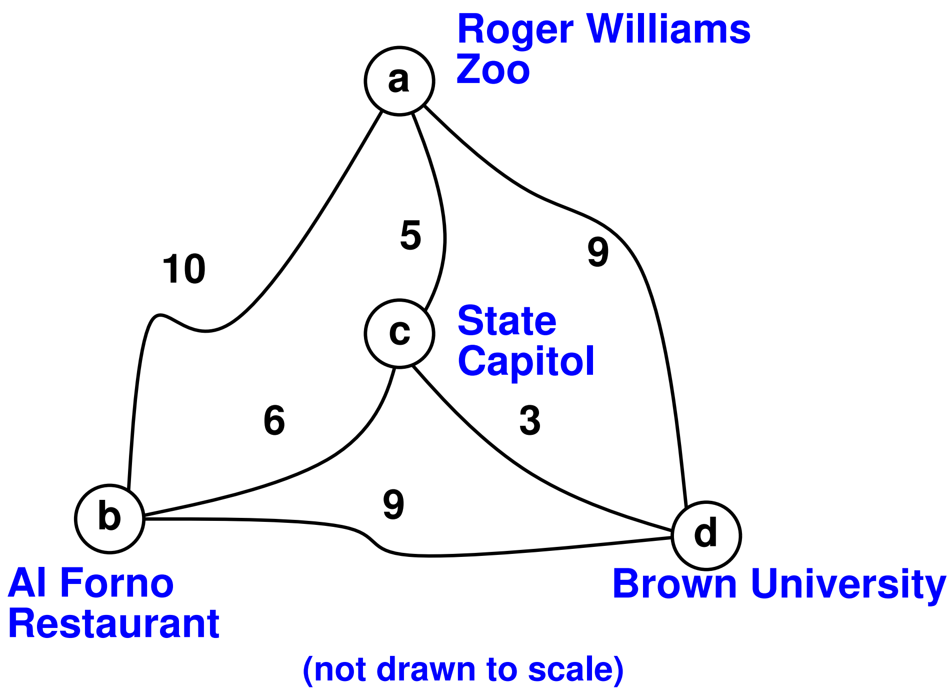
An *algorithm* is a step-by-step procedure

- a recipe
- a computer program
- a mathematical object

We want the most *efficient* algorithm

- fastest (mostly)
- most economical with memory (sometimes)
- expressed as a function of problem size

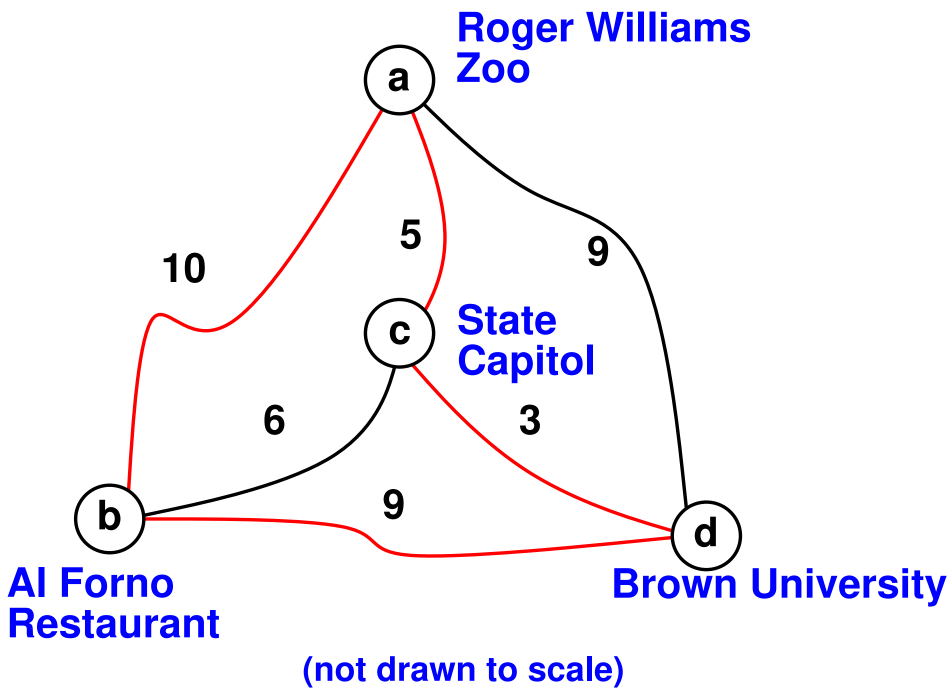
Example: Traveling Salesman Problem



Parameters:

- set of *cities*
- set of inter-city *distances*

Example: Traveling Salesman Problem



Solution:

- want the shortest tour through the cities
- example: a, b, d, c, a has length 27.

Problem Size

What is appropriate measure of problem size?

- m nodes?
- $m(m+1)/2$ distances?

Use an *encoding* of the problem

- alphabet of symbols
- strings: $abcd//10/5/9//6/9//3$.

Measures

- *Problem Size:* length of encoding (here: 32).
- *Time Complexity:* how long an algorithm takes, as function of problem size.

Time Complexity

What is tractable?

A function $f(n)$ is $O(g(n))$ whenever

$$|f(n)| \leq c \cdot |g(n)|$$

for $n >$ some constant.

A *polynomial-time algorithm* is one whose time complexity is $O(p(n))$ for some polynomial $p(n)$.

An *exponential-time algorithm* is one whose time complexity cannot be bounded by a polynomial (e.g., $n^{\log n}$).

Tractability

Basic distinction:

- polynomial time = tractable
- exponential time = intractable

Execution time in microseconds (from Garey and Johnson)

	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.00001 second	.00004 second	.00009 second	.00016 second	.00025 second	.00036 second
n^3	.00001 second	.00008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minute	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	$2 \cdot 10^8$ centuries	$1.3 \cdot 10^{13}$ centuries

Effect of Speed-Ups

Let's wait for faster hardware!

Consider maximum problem size you can solve in an hour.

	present	100 times faster	1000 times faster
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
n^5	N_4	$2.5N_4$	$3.98N_4$
2^n	N_5	$N_5 + 6.64$	$N_5 + 9.97$
3^n	N_6	$N_6 + 4.19$	$N_6 + 6.29$

Your boss says:

> "Get me an efficient traveling-salesman algorithm, or else."

What are you going to do?

Response

"Yes Ma'am, expect it this afternoon!"

Problem is

- All known algorithms (essentially) check all possible paths.
- Exhaustive checking is exponential.
- Good luck!

Response

"Hah!, I will prove that no such algorithm is possible"

Problem is, proving intractability is very hard.

Many important problems have

- no known tractable algorithms
- no known proof of intractability.

Response

"I can't find an efficient algorithm, I guess I'm just a pathetic loser. "

Bad for job security.

Response

"The problem is NP-complete. I can't find an efficient algorithm, but neither can any of these famous people . . ."

Advantage is:

- The problem is "just as hard" as other problems smart people can't solve efficiently.
- So it would do no good to fire you and hire a Dartmouth graduate.

Response

"Would you settle for a pretty good algorithm?"

Intractability isn't everything.

- find an *approximate* solution
- use randomization ("probably pretty good")
- parallelism can also help

Approximation, randomization, etc. are (arguably) the hottest areas in complexity theory today.

Computability

In the first half of the 20th century, mathematicians such as Kurt Göedel, Alan Turing, and Alonzo Church discovered that some problems do not have a computable solution.

Example: is a mathematical statement true or false?

What could be more amenable to automation?

Results needed theoretical models for computers.

These theoretical models helped lead to the construction of real computers.

Computability

Hilbert's tenth problem (1900): Find an finite procedure (algorithm) that decides whether a polynomial has an integer root.

A *polynomial* is a sum of terms, each term is a product of variables and a constant (called coefficient).

A *root* of a polynomial is an assignment of values to the variables such that the value of the polynomial is 0.

$x = 2, y = 1$ is a root of the polynomial $5x + 15y = 25$

There is *no* such algorithm (1970)!

Some Other Undecidable Problems

- Does a program run forever?
- Is a program correct?
- Are two programs equivalent?
- Is a program optimal?
- Is a finitely-presented group trivial?