Week 13 - Monday

COMP 4500

Last time

- What did we talk about last time?
- Exam 3 post mortem
- Finished Co-NP
- A little theory of computing

Questions?

Assignment 7

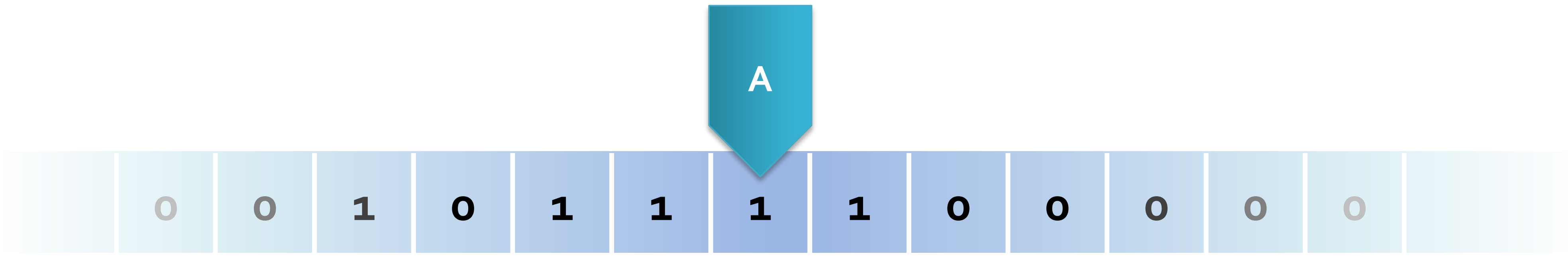
Logical warmup

- You are the despotic ruler of an ancient empire
- Tomorrow is the 25th anniversary of your reign
- You have 1,000 bottles of wine you were planning to open for the celebration
- Your Grand Vizier uncovered a plot to murder you:
  - 10 years ago, a rebel worked in the vineyard that makes your wine
  - He poisoned one of the 1,000 bottles you are going to serve tonight and has been waiting for revenge
  - The rebel died an accidental death, and his papers revealed his plan, but not which bottle was poisoned
- The poison exhibits no symptoms until death
- Death occurs within ten to twenty hours after consuming even the tiniest amount of poison
- You have over a thousand slaves at your disposal and just under 24 hours to determine which single bottle is poisoned
- You have a few hundred prisoners, sentenced to death
- Can you use just the prisoners and risk no slaves?
- If so, what's the smallest number of prisoners you can risk and still be sure to find the bottle?

Computability

Turing machine

- A Turing machine is a mathematical model for computation
- It consists of a head, an infinitely long tape, a set of possible states, and an alphabet of characters that can be written on the tape
- A list of rules saying what it should write and should it move left or right given the current symbol and state



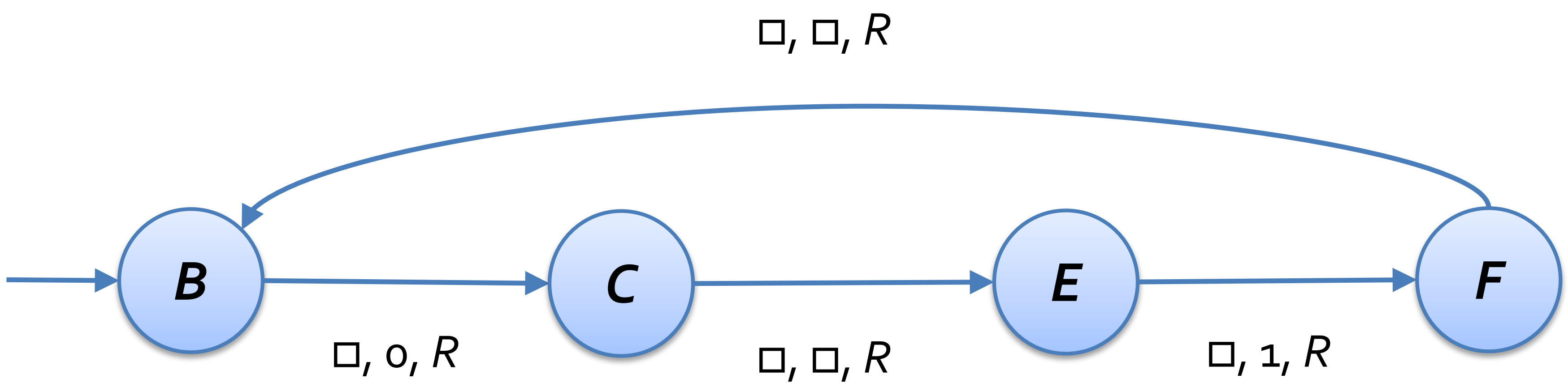
Turing machine example

- You can specify a Turing machine with a table giving its behavior for a specific configuration
- Turing's first example machine printed an infinite sequence of alternating 1s and 0s, separated by spaces:

Configuration		Behavior	
State	**Symbol**	**Operation**	**Result State**
B	Blank	Write 0, Move Right	C
C	Blank	Write Blank, Move Right	E
E	Blank	Write 1, Move Right	F
F	Blank	Write Blank, Move Right	B

A Turing machine as a transition diagram

- The transition table from the previous slide can be drawn as a transition diagram too:



Church-Turing thesis

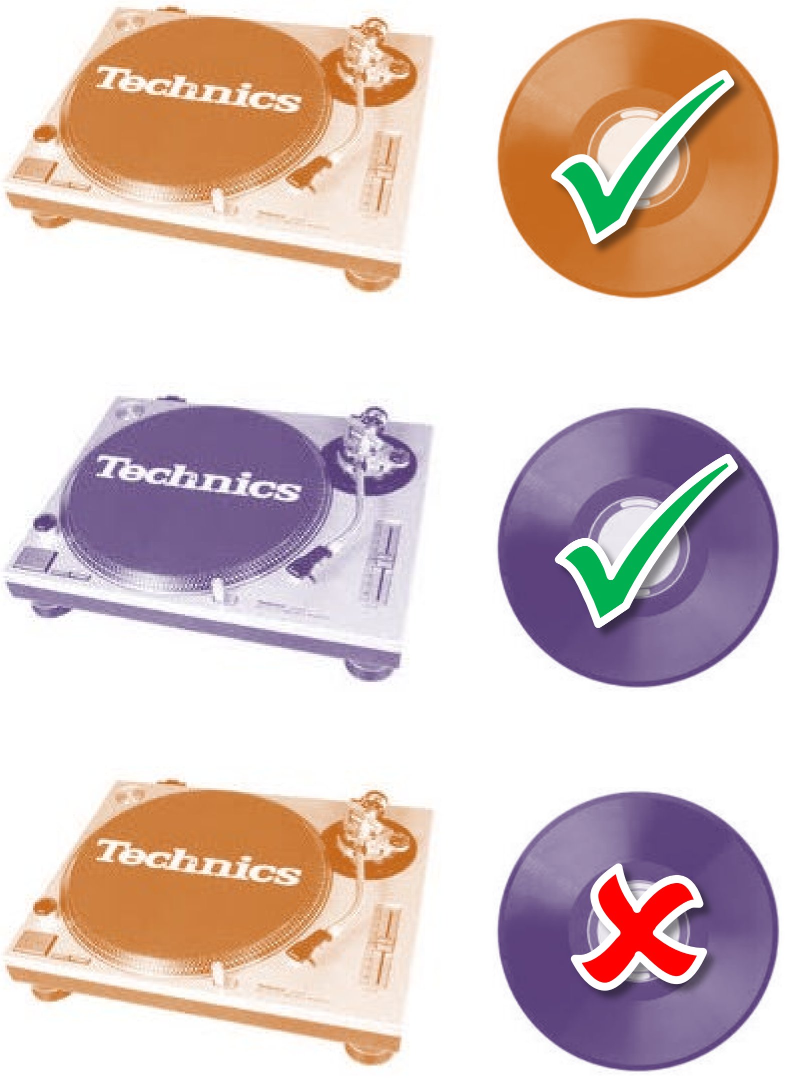
- If an algorithm exists, a Turing machine can perform that algorithm
- In essence, a Turing machine is the most powerful model we have of computation
- Power, in this sense, means the ***ability*** to compute some function, **not** the ***speed*** associated with its computation
- Do you own a Turing machine?

Halting problem

- Given a Turing machine and input ***x***, does it reach the halt state?
- First, recognize that we can encode a Turing machine as input for another Turing machine
  - We just have to design a system to describe the rules, the states, etc.
- We want to design a Turing machine that can read another

Turntables

- Douglas Hofstadter uses the metaphor of turntables
- Imagine that evil people design records that will shake turntables apart when they're played
- Maybe turntable A can play record A and turntable B can play record B
- However, if turntable A plays record B, it will shatter

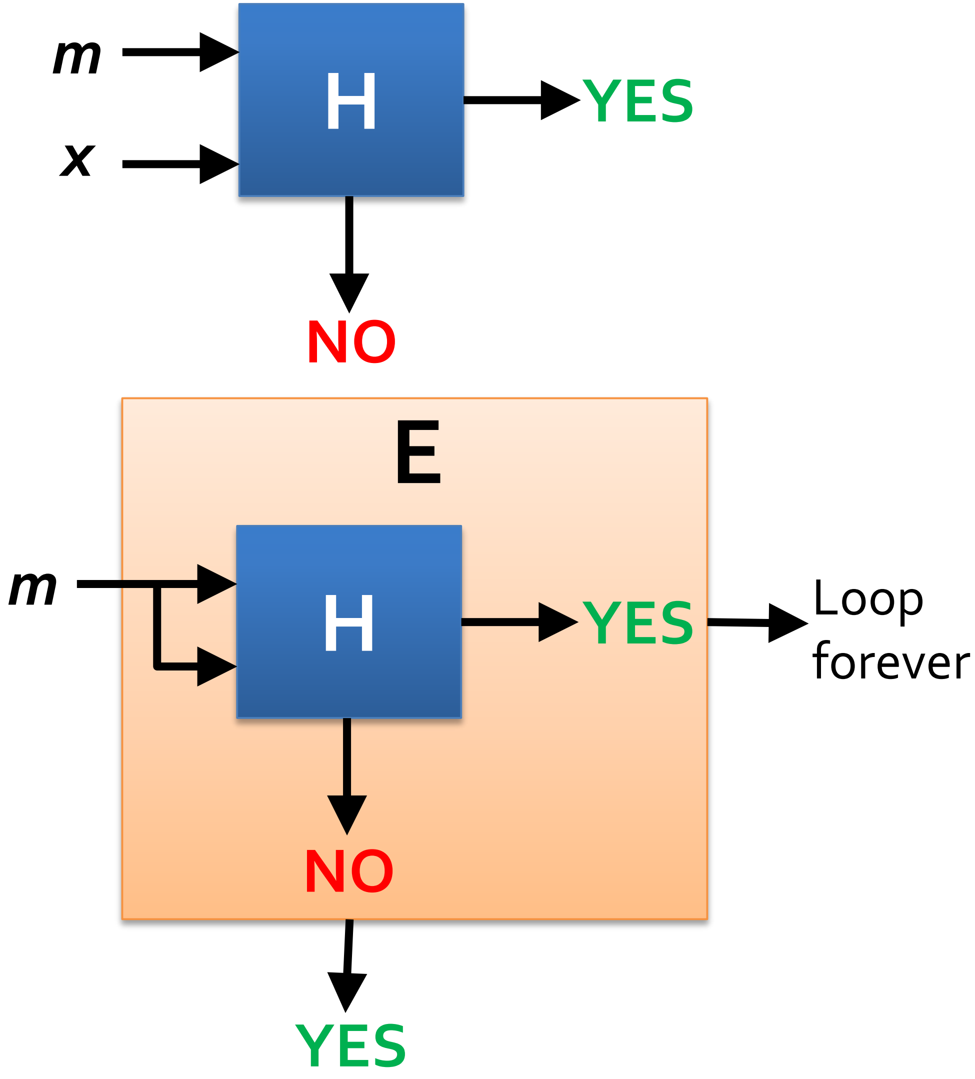


Stuff you have to buy for this proof

- Turing machines can perform all possible computations
- It's possible to encode the way a Turing machine works such that another Turing machine can read it
- It's easy to make a slight change to a Turing machine so that it gives back the opposite answer (or goes into an infinite loop)

Proof by contradiction

- You've got a Turing machine **M** with encoding ***m***
- You want to see if **M** will halt on input ***x***
- Assume there is a machine **H** that can take encoding ***m*** and input ***x***
  - **H**(***m***,***x***) is YES if it halts
  - **H**(***m***,***x***) is NO if it loops forever
- We create (evil) machine **E** that takes description ***m*** and runs **H**(***m***,***m***)
  - If **H**(***m***,***m***) is YES, **E** loops forever
  - If **H**(***m***,***m***) is NO, **E** returns YES



A mind-bending proof

- Let's say that ***e*** is the description of **E**
- What happens if you feed description ***e*** into **E**?
  - **E**(***e***) says what **E** will do with itself as input
- If it returns **YES**, that means that **E** on input ***e*** loops forever
  - But it can't, because it just returned **YES**
- If it loops forever, then **E** on input ***e*** would return **YES**
  - But it can't, because it's looping forever!
- Our assumption that machine **H** exists must be wrong

Halting problem conclusion

- Clearly, a Turing machine that solves the halting problem doesn't exist
- Essentially, the problem of deciding if a problem is computable is itself uncomputable
- Therefore, there are some problems (called **undecidable)** for which there is no algorithm
- Not an algorithm that will take a long time, but **no algorithm**
- If we find such a problem, we are stuck
- … unless someone can invent a more powerful model of computation

Post correspondence problem

- Given two finite lists of words A and B, can you pick ***k*** words (repetitions allowed) from A and ***k*** words (repetitions allowed) from B so that the words from A concatenated are exactly the same string as the words from B concatenated
- Example:

A	B
aa	bbbb
aba	a
bb	abba

- Solution:
  - aa + bb + bb = aabbbb = a + a + bbbb
- The Post correspondence problem (PCP) is **undecidable** (there is no algorithm that can solve all instances of it)

Other undecidable problems

- Are two context-free languages the same?
- Is the intersection of two context-free languages empty?
- Is a context-free language equal to Σ^*?
- Is a context-free language a subset of another context-free language?
- Is a given statement of first-order logic provable from a starting set of axioms?
- Given a set of matrices, is there some sequence that they can be multiplied in (perhaps with repetitions) that will yield the zero matrix?

More (useful) undecidable problems

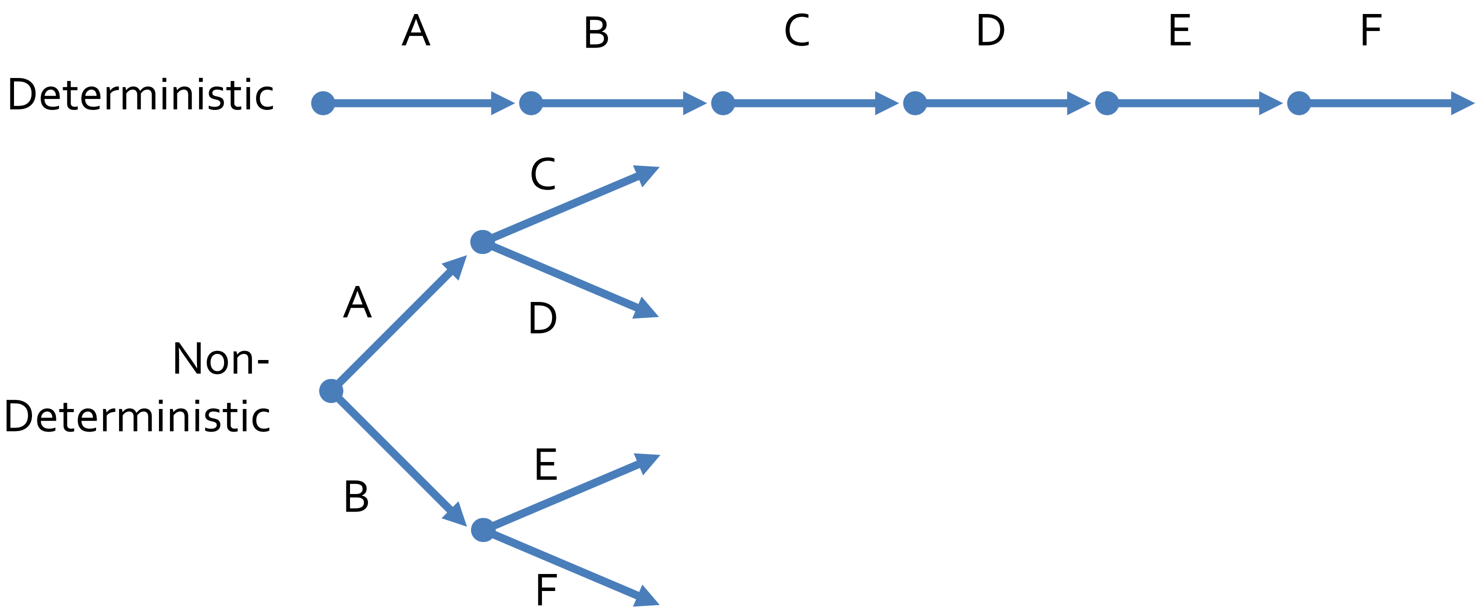
- Does a given Java program have an infinite loop in it?
- Will a given Python program terminate regardless of what inputs it's given?
- Will computation with input ***x*** use all states of a Turing machine?
- Given input ***x***, will the output of a C++ program be ***y***?

NP

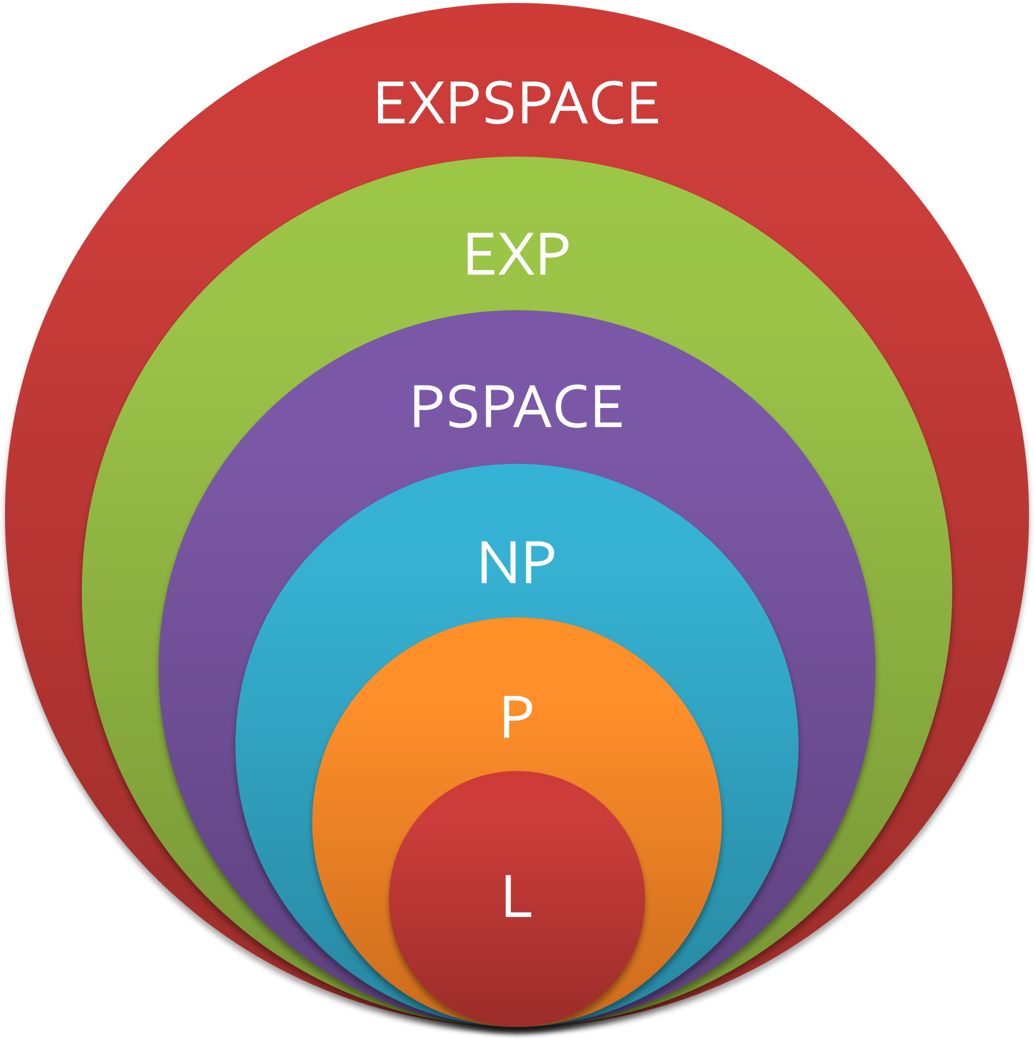
- Problems in NP can be solved in polynomial time on a non-deterministic computer
- A deterministic computer is the kind you know:
  - First it has to consider possibility A, then, it can consider possibility B

Deterministic vs. non-deterministic

- A non-deterministic computer (which, as far as we know, only exists in our imagination) can consider both possibility A and possibility B at the same time

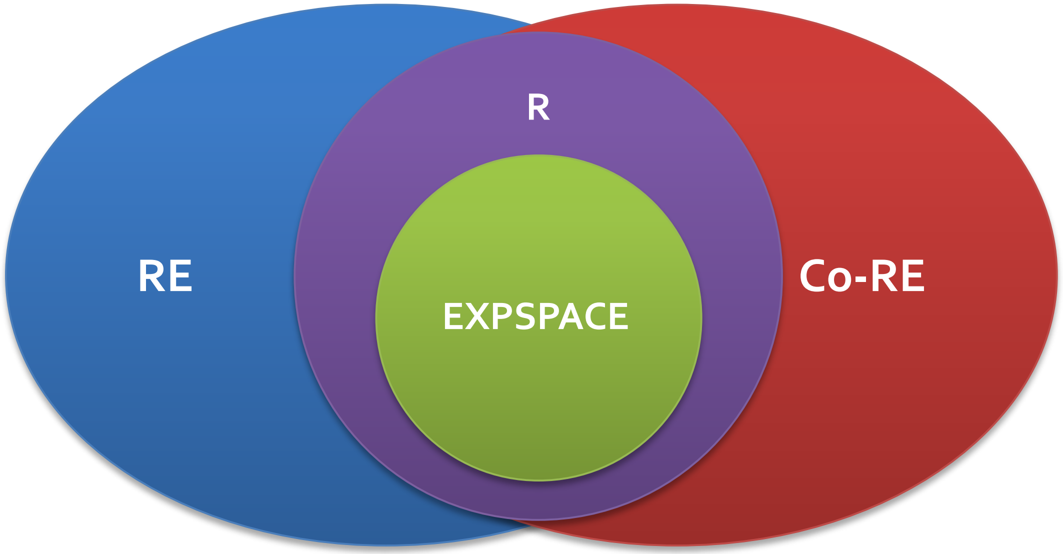


Complexity hierarchy



- EXPSPACE
  - Exponential space
- EXP
  - Exponential time on a deterministic Turing machine
- PSPACE
  - Polynomial space
- NP
  - Polynomial time on a non-deterministic Turing machine
- P
  - Polynomial time on a deterministic Turing machine
- L
  - Logarithmic space on a deterministic Turing machine

Going beyond running time



- **R** is the set of recursive decision problems
- **RE** is the set of recursively enumerable decision problems
- **Co-RE** is the complement of set **RE**

- There are also problems that cannot be solved by any computer (or algorithm) in a finite amount of time
- Problems in **R** can be solved by an algorithm in a finite amount of time
- Problems in **RE** can reach a "yes" answer in a finite amount of time, but no algorithm is guaranteed to reach completion if the answer is "no"
- Problems in **Co-RE** can reach a "no" answer in a finite amount of time, but no algorithm is guaranteed to reach completion if the answer is "yes"

Three-Sentence Summary of Approximation Algorithms for Load Balancing and Center Selection

Approximation Algorithms

Approximation algorithms

- What can you do when faced with an NP-complete problem?
  - Or, more likely, an NP-hard problem, where you come up with the optimal answer, not just a "yes" or a "no"
- One possibility is using an approximation algorithm
  - You don't get the guarantee of the perfect optimal answer
  - But you might be able to get a reasonably good answer in polynomial time

Load balancing

- You have m machines $M_1, M_2, \ldots, M_m$
- You have n jobs
- Each job j has a processing time t_j
- We can assign jobs $A(i)$ to machine M_i
- The total time that M_i needs to work is:

$$T_i = \sum_{j \in A(i)} t_j$$

- We want to minimize the **makespan**, which is just the longest T_i
- In other words, we want the last machine running to stop running as early as possible
- Unfortunately, doing so in **NP-hard**

Greedy algorithm

- We can use a simple greedy algorithm for assigning jobs:
  - For each job ***j***, assign it to the machine that has the shortest completion time so far
- Using this algorithm, what would the makespan be for three machines M_1, M_2, and M_3, given the following job sizes:
  - 2, 3, 4, 6, 2, 2
- What would the optimal be?

Lower bound

- Approximation algorithms are often very simple
- The hard part is doing the analysis to show that the result is not too bad relative to optimal
- Can we give a lower bound on how big the optimal makespan ***T****** must be?
- It **cannot** be shorter than the longest job, whatever job that is
- Also, if we perfectly balanced the work among ***m*** machines, each machine would still have to do at least $\frac{1}{m}$ of the total work

Greedy algorithm gets a makespan $T \leq 2T^*$

- **Proof:**
  - Let $\boldsymbol{M_i}$ be the machine that get the maximum load $\boldsymbol{T}$ in the greedy assignment
  - Let $\boldsymbol{j}$ be the last job assigned to $\boldsymbol{M_i}$
  - When $\boldsymbol{j}$ was assigned to $\boldsymbol{M_i}$, it had the smallest load of any machine, namely $\boldsymbol{T_i - t_j}$
  - Thus, every machine had load at least $\boldsymbol{T_i - t_j}$

$$\sum_{k=1}^{m} T_k \geq m\left(T_i - t_j\right)$$

$$T_i - t_j \leq \frac{1}{m} \sum_{k=1}^{m} T_k$$

Proof continued

- Since $\sum_{k=1}^{m} T_k = \sum_{i=1}^{n} j_i$ and $\frac{1}{m}\sum_{i=1}^{n} j_i \leq T^*$

$$T_i - t_j \leq T^*$$

- But the optimal makespan must be at least as big as any job, thus $t_j \leq T^*$, thus:

$$T_i = \left(T_i - t_j\right) + \ t_j \leq T^* + T^* = 2T^*$$

- Since our makespan $T = T_i$, the proof is done.

■

Upcoming

Next time...

- Finish load balancing
- Center selection
- Set cover
- Read section 11.3

Reminders

- Start Assignment 7
- Office hours from 2-4 p.m. today canceled for the eclipse