Week 11 - Wednesday

COMP 2100

Last time

- What did we talk about last time?
- Finished B-trees
- Hard problems on graphs

Questions?

Project 3

Assignment 6

Just in case you want something else to work on …

Hard Problems

Knapsack problem

- Not all NP-complete problems are graph problems
- The knapsack problem is the following:
  - Imagine that you are Indiana Jones
  - You are the first to open the tomb of some long-lost pharaoh
  - You have a knapsack that can hold ***m*** pounds of loot, but there's way more than that in the tomb
  - Because you're Indiana Jones, you can instantly tell how much everything weighs and how valuable it is
  - You want to find the most valuable loot that weighs less than or equal to ***m*** pounds

Subset sum

- This one is a little bit mathematical
- Say you have a set of numbers
- Somebody gives you a number ***k***
  - Is there any subset of the numbers in your set that add up to exactly ***k***?
- Example:
  - Set: { 3, 9, 15, 22, 35, 52, 78, 141}
  - Is there a subset that adds up to exactly 100?
  - What about 101?

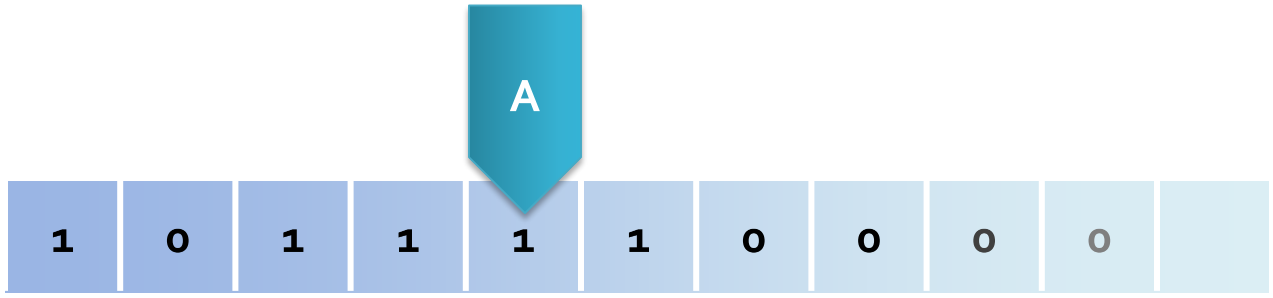
Fabulous Cash Prizes

- These NP-complete problems are very hard
- Many of them are really useful
  - Especially if you are a lazy traveling salesman
- Clay Mathematics Institute has offered a **$1,000,000** prize
- You can do one of two things to collect it:
  - Find an efficient solution to any of the problems
  - Prove that one cannot have an efficient solution

A Few Finer Points…

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

- 3 state, 2 symbol "busy beaver" Turing machine:

Tape Symbol	State A			State B			State C		
	Write	Move	Next	Write	Move	Next	Write	Move	Next
0	1	R	B	0	R	C	1	L	C
1	1	R	HALT	1	R	B	1	L	A

- Starting state A

Church-Turing thesis

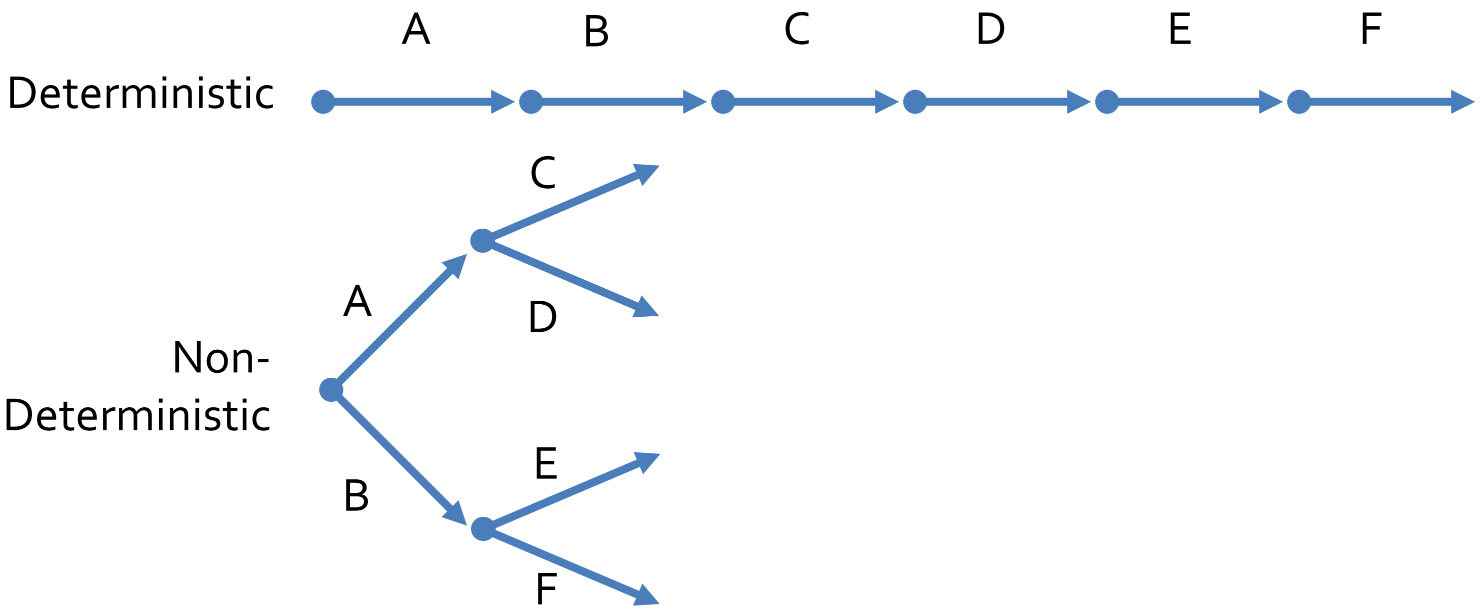
- If an algorithm exists, a Turing machine can perform that algorithm
- In essence, a Turing machine is the most powerful model we have of computation
- Power, in this sense, means the ***ability*** to compute some function, **not** the ***speed*** associated with its computation

NP

- NP is actually a class of problem
- 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
- It's like a computer that can keep spawning threads and always has a core to execute a new thread on



P

- P is the class of decision problems that can be solved in polynomial time by a deterministic computer
- Lots of great problems are in P:
  - Is this list sorted?
  - Is this number prime?
  - Is the largest number in this B-tree equal to 38?
- Many problems are unknown:
  - Does this number have exactly two factors?
  - Is this graph equivalent to this other graph?

NP-complete

- Everything in P is also in NP
- Some problems are the "hardest" problems in NP
- This means that any problem in NP can be converted into one of these problem in polynomial time
- These problems make up the class **NP-complete**

Decisions, decisions

- Notice that P, NP, and NP-complete are all decision problems
- So, the TSP we stated is not technically NP-complete
- The NP-complete version is:
  - Is there a tour of length less than or equal to 24 in this graph?
- The optimization versions of NP-complete problems are called NP-hard

Easy to check vs. easy to answer

- Computer scientists view problems in P as "easy to answer"
  - They can be computed in polynomial time
- Problems in NP are "easy to check"
  - An answer can be checked in polynomial time
- For example, if someone gives you a Traveling Salesman tour, you can verify that it is a legal tour of the required length
- But is easy to check the same as easy to answer?

What if P = NP?

- Most computer scientists think that P ≠ NP
- But if it were
  - Most things could be perfectly scheduled
    - e.g., the best room for a given number of students and the time preferences of everyone involved
  - All routing and path planning (UPS, military, etc.) would be optimal
  - It might be possible to devise perfect genetic therapies for certain conditions
  - It would be possible to prove all kinds of previously unproven theorems in mathematics

What if P = NP? (The Bad)

- On the other hand, if P = NP, it might also mean:
  - Most of our encryption algorithms would be broken
  - All computer, Internet, and banking security would be worthless
- Could creativity be doomed?
  - If recognizing something good was the same as creating something good … who knows?

A final word

> *If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett. It's possible to put the point in Darwinian terms: if this is the sort of universe we inhabited, why wouldn't we already have evolved to take advantage of it?*
>
> Scott Aaronson

Review

Recursion

Recursion

- Base case
  - Tells recursion when to stop
  - Can have multiple base cases
  - Have to have at least one or the recursion will never end
- Recursive case
  - Tells recursion how to proceed one more step
  - Necessary to make recursion able to progress
  - Multiple recursive cases are possible

Recursive function example

- Factorial:

```
public static int factorial(int n) {
  if( n == 1 ) {
     return 1;
  } else {
     return n * factorial(n - 1);
  }
}
```

Trees

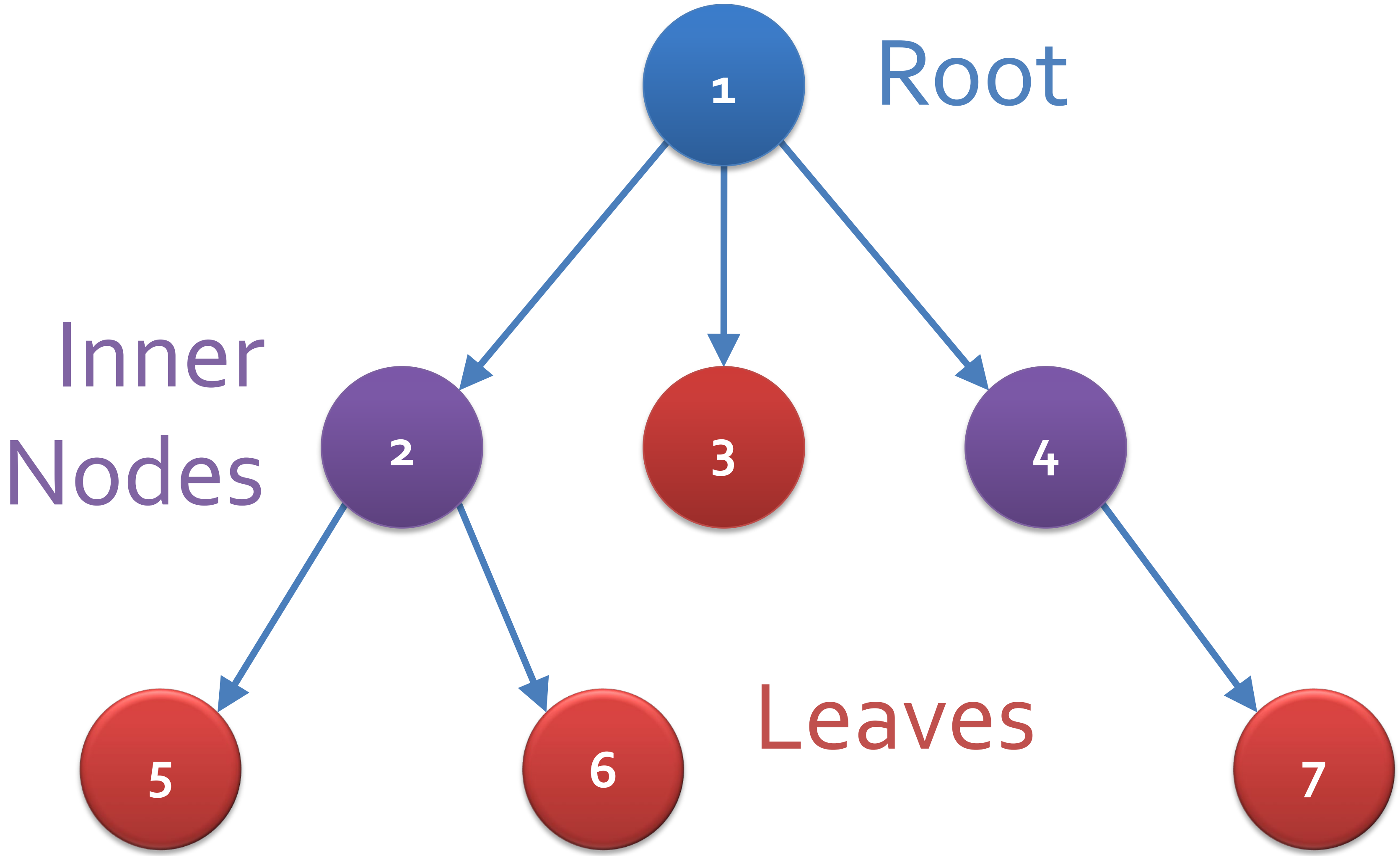
Trees

- A tree is a data structure built out of nodes with children
- A general tree node can have any non-negative number of children
- Every child has exactly one parent node
- There are no loops in a tree
- A tree expressions a hierarchy or a similar relationship

Terminology

- The **root** is the top of the tree, the node which has no parents
- A **leaf** of a tree is a node that has no children
- An **inner node** is a node that does have children
- An **edge** or a **link** connects a node to its children
- The **depth** of a node is the length of the path from a node to its root
- The **height** of the tree is the greatest depth of any node
- A **subtree** is a node in a tree and all of its children
- **Level:** the set of all nodes at a given depth from the root

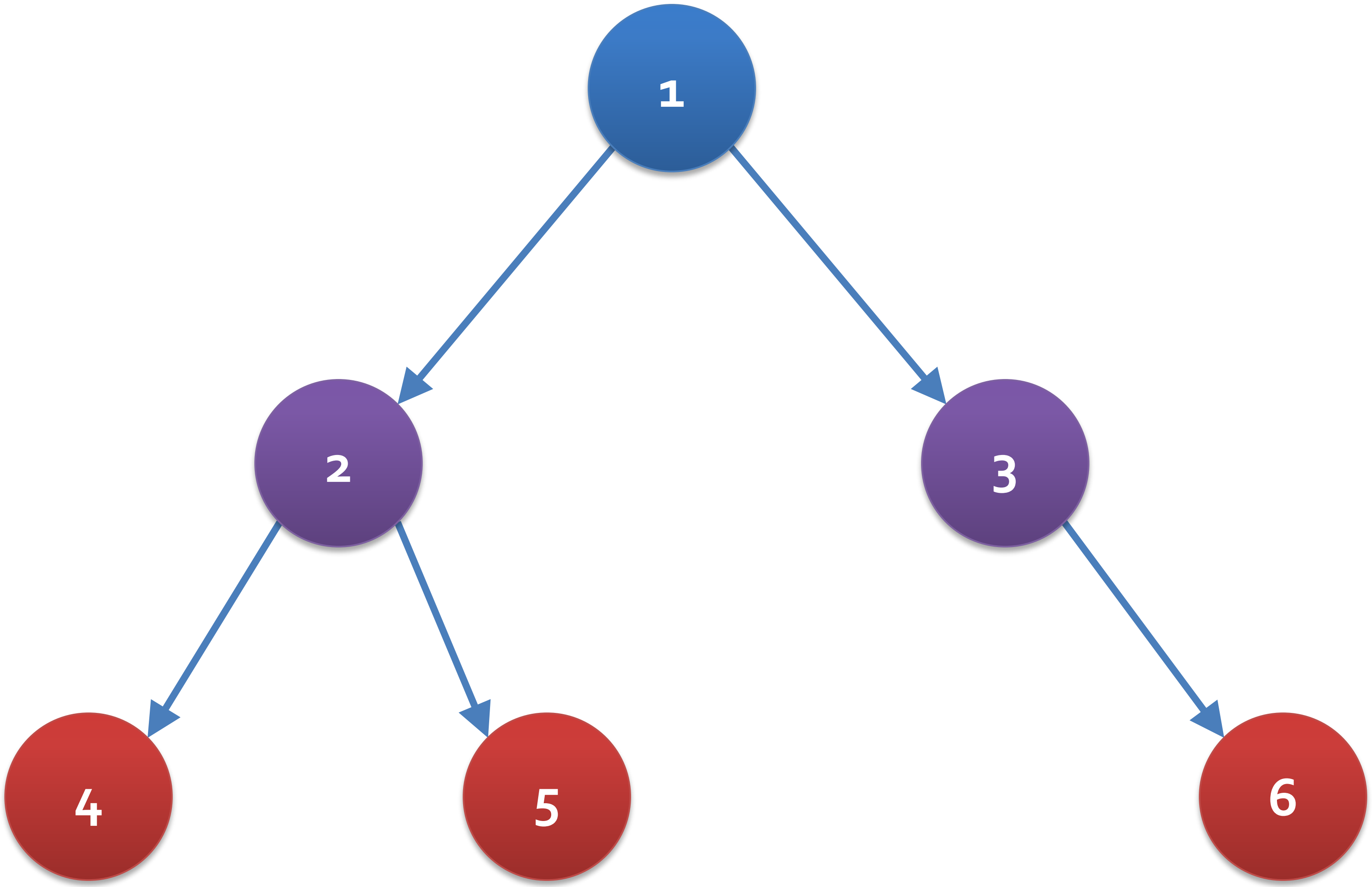
A tree



Binary tree

- A binary tree is a tree such that each node has two or fewer children
- The two children of a node are generally called the **left child** and the **right child**, respectively

Binary tree



Binary tree terminology

- **Full binary tree:** every node other than the leaves has two children
- **Perfect binary tree:** a full binary tree where all leaves are at the same depth
- **Complete binary tree:** every level, except possibly the last, is completely filled, with all nodes to the left
- **Balanced binary tree:** the depths of all the leaves differ by at most 1

Quiz

Upcoming

Next time...

- Review

Reminders

- **4-5 p.m. office hours canceled today because of Faculty Assembly**
- Finish Project 3
  - Due Friday by midnight
- Review chapters 3 and 4 for Exam 2
  - **Next Monday!**
  - We'll review more for Exam 2 on Friday