Announcement

Programming assignment #4 has been posted in Blackboard and course website

Due at 11:59pm, Sunday, April 24th

Announcement

According to the UofSC final exam schedule Final Exam Schedule Spring 2022 - University Registrar | University of South Carolina

Final exam: **May 3, Tuesday, 9:00am – 11:30 am**

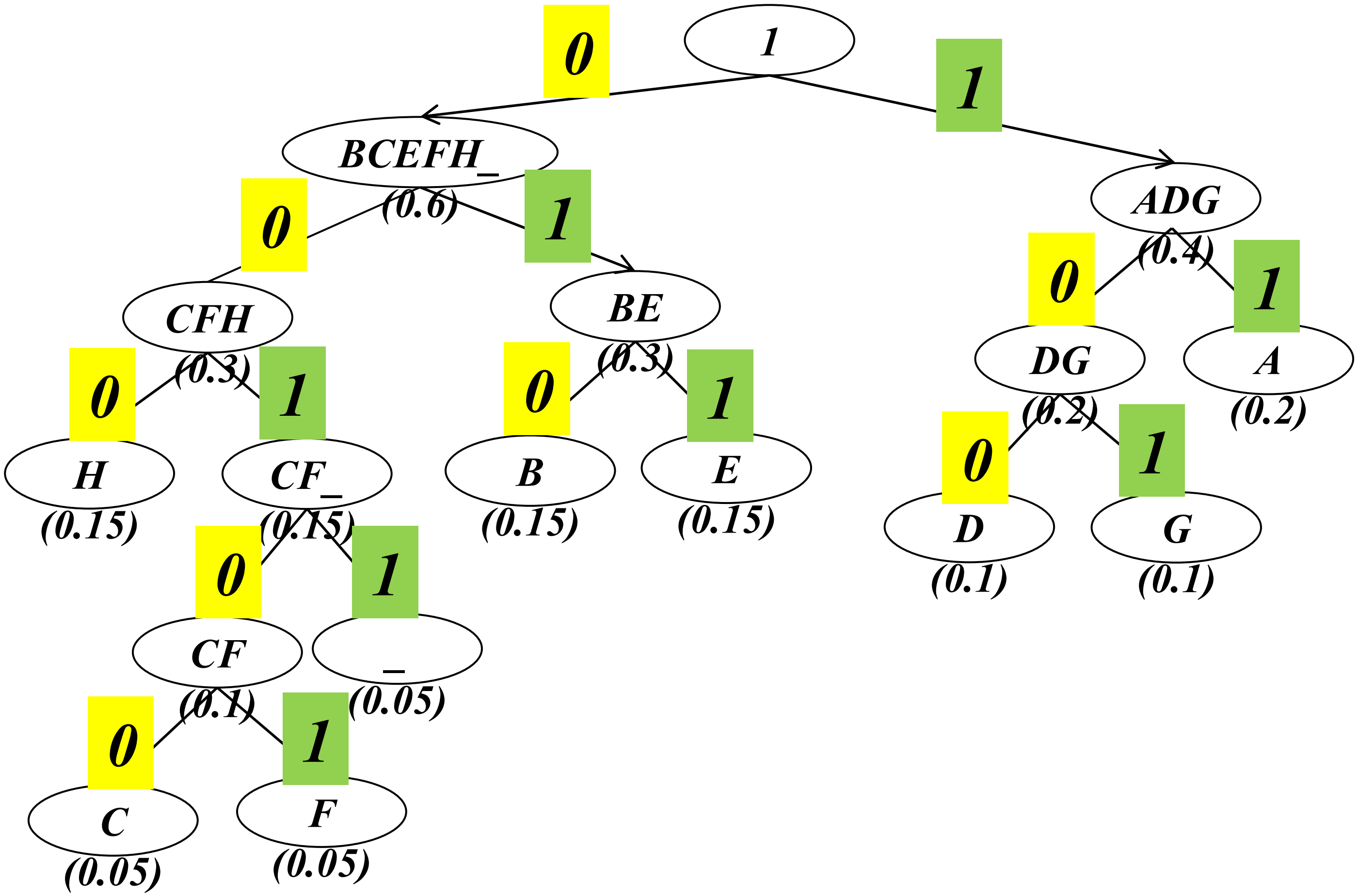
Cover all materials in our lectures

Closed-book and closed-notes.

A double-sided letter-size cheat sheet is allowed

Huffman Coding Example

Character	A	B	C	D	E	F	G	H	_
Probability	0.2	0.15	0.05	0.1	0.15	0.05	0.1	0.15	0.05



1
0
1
BCEFH_
(0.6)
ADG
(0.4)
0
1
0
1
CFH
(0.3)
BE
(0.3)
DG
(0.2)
A
(0.2)
0
1
0
1
0
1
H
(0.15)
CF_
(0.15)
B
(0.15)
E
(0.15)
D
(0.1)
G
(0.1)
0
1
CF
(0.1)
_
(0.05)
0
1
C
(0.05)
F
(0.05)

Character	A	B	C	D	E	F	G	H	_
Probability	0.2	0.15	0.05	0.1	0.15	0.05	0.1	0.15	0.05
Codeword	11	010	00100	100	011	00101	101	000	0011
Code length	2	3	5	3	3	5	3	3	4

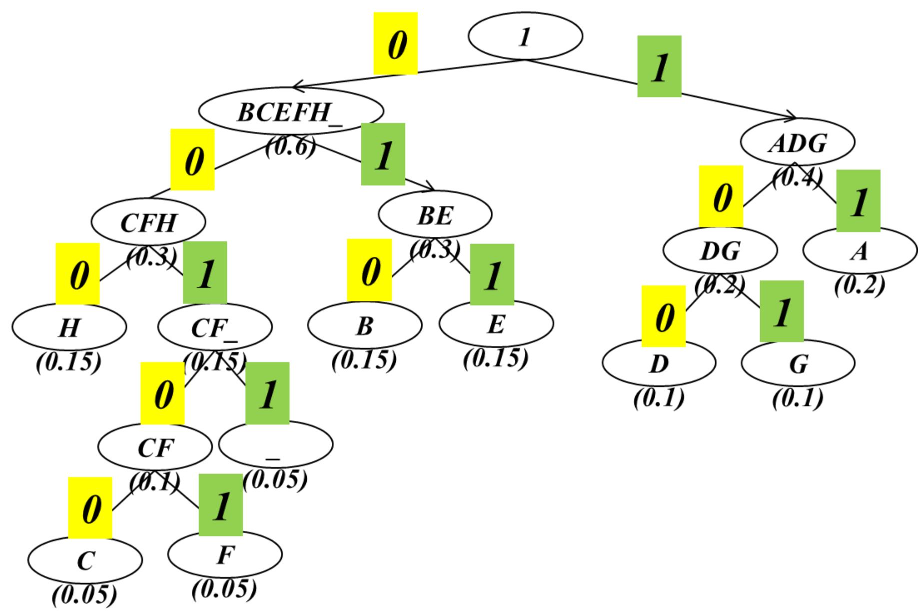
Average number of bits per character (code length):

$$0.2 * 2 + 0.15 * 3 + 0.05 * 5 + 0.1 * 3 + 0.15 * 3 + 0.05 * 5 + 0.1 * 3 + 0.15 * 3 + 0.05 * 4 = 3.05$$

Note:

The resulting Huffman tree varies according to your choices, e.g., assigning 0/1 to left/right

But the average code length is the same



Reading Assignments

Read Chapter 10. Iterative Improvement

Chapter 11: Limitations of Algorithm Power

Basic Asymptotic Efficiency Classes (big O, big Θ, and big Ω)

		n=10	n=100
1	constant	constant	constant
log *n*	logarithmic	1 (with base 10)	2 (with base 10)
n	linear	10	100
n log *n*	*n* log *n*	*10* (with base 10)	*200* (with base 10)
n^2	quadratic	100	10,000
n^3	cubic	1000	1,000,000
2^n	exponential	1024	~$1.26*10^{30}$
n!	factorial	3,628,800	~$9.33*10^{157}$

Polynomial-Time Complexity

Polynomial-time complexity: the complexity of an algorithm is

$$a_b n^b + a_{b-1} n^{b-1} + \ldots + a_1 n^1 + a_0 \Rightarrow \Theta(n^b)$$

with a fixed degree *b*>0. Usually b<=3

If a problem can be solved in polynomial time, it is usually considered to be theoretically tractable in current computers.

When an algorithm's complexity is larger than polynomial, i.e., exponential, theoretically it is considered to be too expensive to be useful – intractable

Polynomial-Time Complexity

Polynomial time complexity

1	constant
log *n*	logarithmic
n	linear
n log *n*	*n* log *n*
n^2	quadratic
n^3	cubic
2^n	exponential
n!	factorial

List of Problems

Sorting $O(nlogn)$

Searching $O(n)$

All shortest paths in a graph $O(|V|^{\mathbf{3}})$

Minimum spanning tree $O(|E|\log|V|)$

Assignment problem $O(n!)\sim O(\boldsymbol{n}^{\mathbf{3}})$

Towers of Hanoi $O(2^n)$

Knapsack problem $O(2^n)$

Traveling salesman problem $\boldsymbol{O}(n!)\sim O\left(\boldsymbol{n}^{\mathbf{2}}2^n\right)$ **–** Current record 85,900 cities (Applegate et al. 2006)
http://en.wikipedia.org/wiki/Travelling_salesman_problem#Computational_complexity

…

Lower Bound

Problem A can be solved by algorithms a_1, a_2,…, a_p

Problem B can be solved by algorithms b_1, b_2,…, b_q

We may ask

- Which algorithm is more efficient? This makes more sense when the compared algorithms solve the same problem
  - It's not fair to compare selection sorting with Warshall's algorithm
- Which problem is more complex? We may compare the complexity of the best algorithm for **A** and the best algorithm for **B**

For each problem, we want to know the lower bound: the best possible algorithm's efficiency for a problem → $\Omega(.)$

Tight lower bound: we have found an algorithm in the this lower-bound efficiency class $\Theta(.)$. The constant factor makes the difference.

Trivial Lower Bound

Many problems need to 'read' all the necessary items and write the 'output'

→ Their sizes provide a trivial lower bound

Example:

1. Generate all permutations of n distinct items → $\Omega(n!)$
   Why?
   Is this a tight lower bound? ***Yes.***
2. Evaluate the polynomial at a given *x*

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

→ $\Omega(n)$, Is this tight? ***Yes.***

Notes on Trivial Lower Bound

Multiplying two *n*x*n* matrices → $\Omega(n^2)$

- because we need to process $2n^2$ elements and output n^2 elements
- We do not know whether this is tight – a lower bound of $\Omega(n^2 logn)$ has been proven Raz 2002

Many trivial lower bounds are too low to be useful

- TSP → $\Omega(n^2)$ because its input is $n(n-1)/2$ intercity distance and output is $n+1$ city in sequence
- There is no known polynomial-time algorithm to solve it

Trivial lower bound sometime have problems

- We do not need to process all the input elements
- For example: searching an element in a sorted array. What is its complexity?

Lower Bound

For each problem, we want to know the lower bound: the best possible algorithm's efficiency for a problem → Ω(.)

Tight lower bound: we have found an algorithm in the this lower-bound efficiency class Θ(.).

Trivial lower bound: the problem's input/output size

- Too low
- Too high

Information-Theoretic Arguments

This approach seeks to establish a lower bound based on the amount of information it has to produce – an information-theoretic lower bound

Recall the problem of guess the number from 1...*n* by asking 'yes/no' questions

Fundamentally, it is a coding problem. If the input number can be encoded into *m* bits, each 'yes/no' question just resolve one bits and therefore, the lower bound is *m*

We know that $m=\log_2 n$

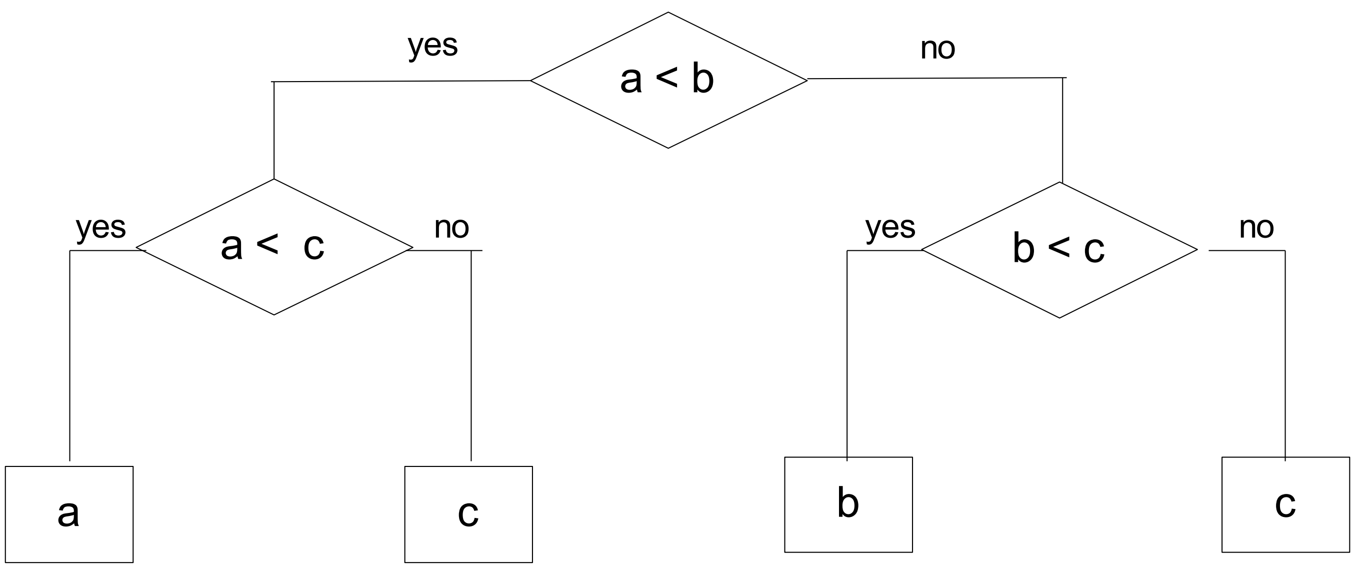
Solution: a decision tree. We will apply the decision tree to find the lower bound for several problems

Find the Smallest from three numbers using comparison

Leaves in the decision tree is the possible output. The output size is at least *3* (maybe larger than *3*) here

Complexity for the worst case: the height of this decision tree

Given *L* leaves, the height of the binary tree is at least $\lceil \log_2 L \rceil$



Decision Tree for Sorting Algorithms

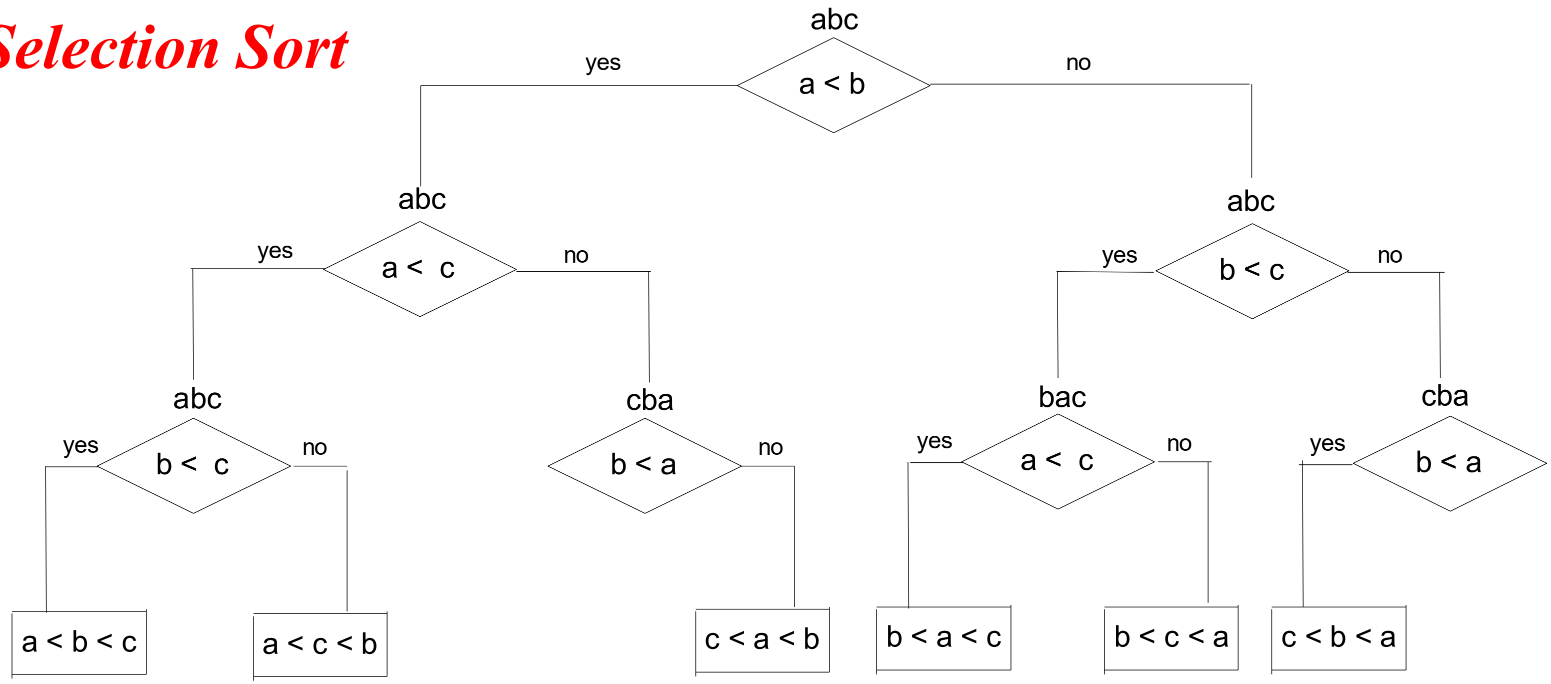
The number of leaves: *n*!

Stirling's

The lower bound for worst case: $\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n \approx n \log_2 n$

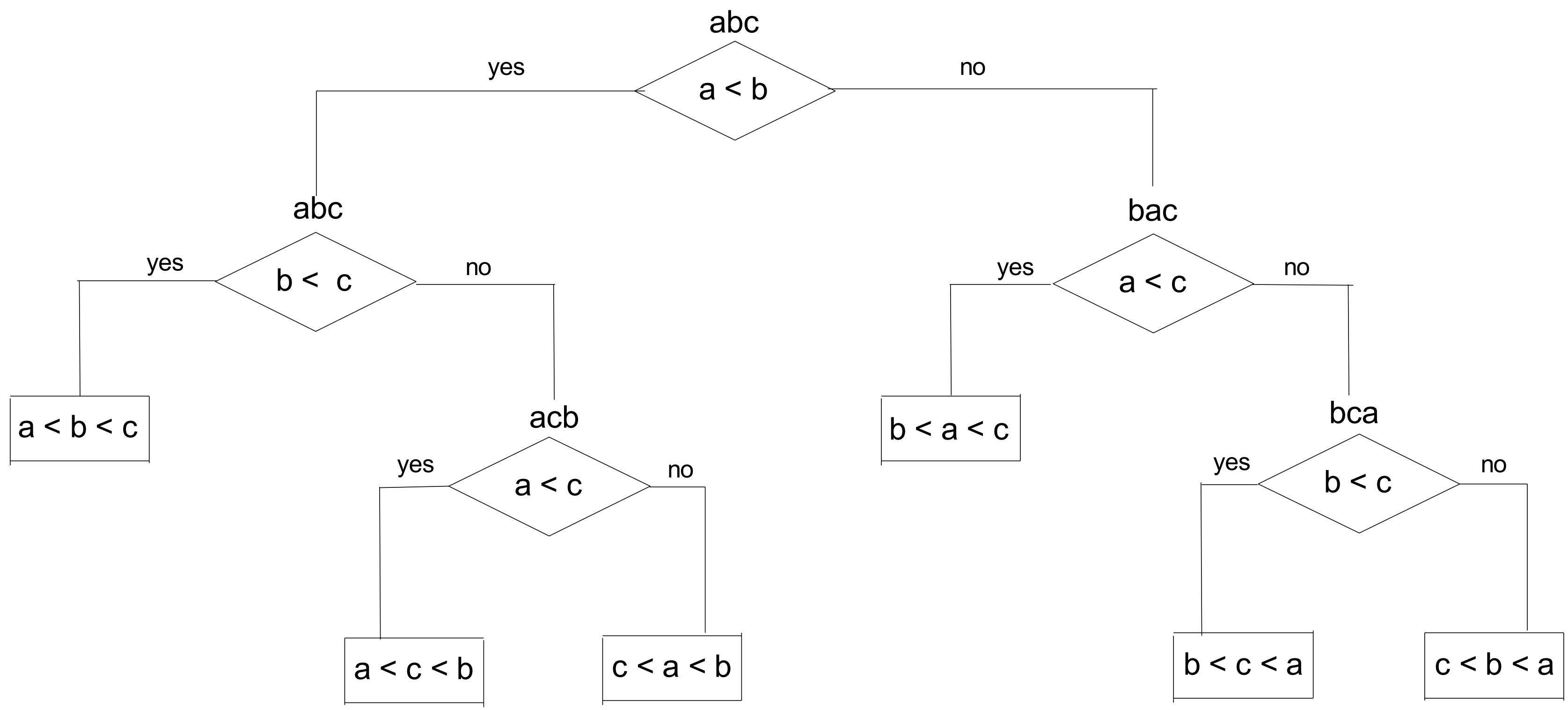
Is this tight?

Selection Sort



Average number of comparisons: (3+3+3+3+3+3)/6 $= 3 = \lceil \log_2 6 \rceil$

Example: Decision Tree for Insertion Sort

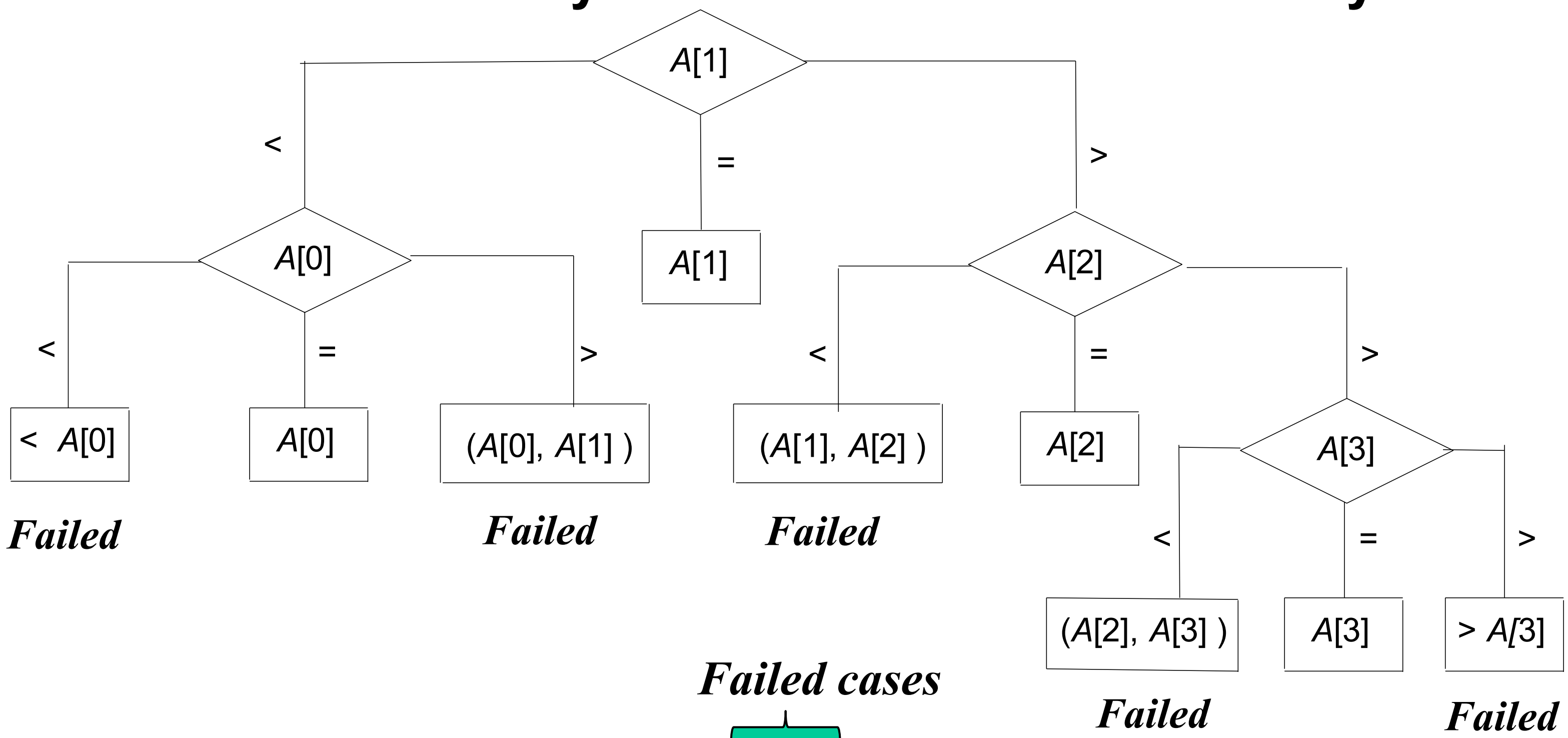


Average number of comparisons: $(2+3+3+2+3+3)/6 \approx 2.666 > \log_2 6$

Worst case: 3 comparisons $= \lceil \log_2 6 \rceil$

Decision Tree for Searching a Sorted Array

Decision tree for binary search in a four-element array

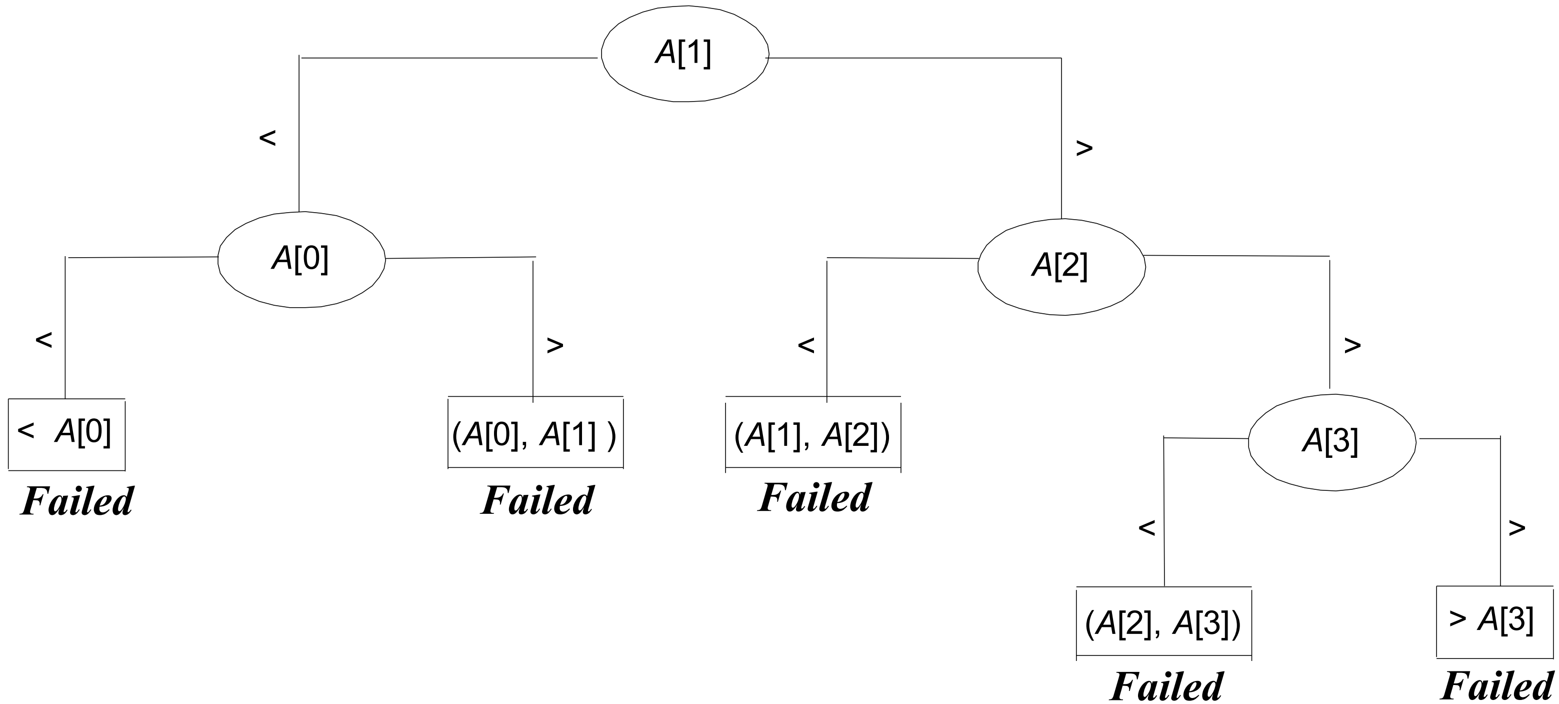


# of leaves for n elements → n+n+1

→lower bound for the worst case: $\lceil \log_3(2n+1) \rceil = \log_3 9 = 3$

→ Ω(logn)

Binary Search → Binary Decision Tree



Lower bound is then $\lceil \log_2(n+1) \rceil$ → **A tight lower bound**

Leaves → unsuccessful search

Parent nodes → successful search

P, *NP*, and *NP*-Complete Problems

As we discussed, problems that can be solved in polynomial time are usually called tractable and the problems that cannot be solved in polynomial time are called intractable, now

Is there a polynomial-time algorithm that solves the problem?

Possible answers:

- yes
- no
  - because it can be proved that all algorithms take exponential time
  - because it can be proved that no algorithm exists at all to solve this problem
- don’t know
- don’t know, but if such algorithm were to be found, then it would provide a means of solving many other problems in polynomial time

Types of Problems

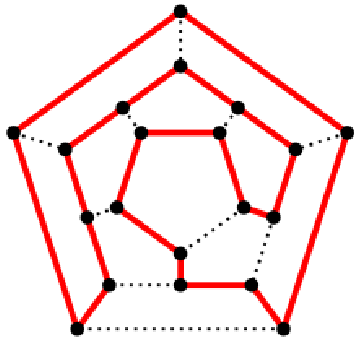
Two types of problems:

- ***Optimization problem:*** **construct a solution that maximizes or minimizes some objective function**
  - MST, all shortest paths, single source shortest paths, …
- ***Decision problem:*** **answer yes/no to a question**
  - Selection, searching, …

Many problems have BOTH decision and optimization versions.

Eg: Traveling Salesman Problem

- *optimization*: find Hamiltonian cycle of minimum weight
- *decision*: Is there a Hamiltonian cycle of weight < *k*



Hamiltonian Circuit*: a closed path in a graph that visits every node in the graph exactly once*

Deterministic VS Nondeterministic Algorithm

A *deterministic algorithm* is the algorithm we discussed before

- A math function: given a specific input, generate the same and unique output in different runs

A *nondeterministic algorithm* is the counterpart

- *May have different outputs in different runs*
- *It is a two-stage process:*
  - *Guessing stage:* generate a random string ***S*** as a candidate solution
  - *Verification stage:* using a ***deterministic*** algorithm which takes the original input ***I*** and ***S*** as input and determine if ***S*** is a solution to ***I***

Why becomes nondeterministic?

- *System noise*
- *random number generator*

Example: Conjunctive Normal Form (CNF) Satisfiability

Problem: Is a Boolean expression in its conjunctive normal form (CNF), i.e., are there "true" or "false" assignments of these variables that makes the Boolean expression true?

This problem is in *NP*.

Nondeterministic algorithm:

- Guess truth assignment
- Check assignment to see if it satisfies CNF formula

Example: (Boolean operation)

$$(a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b} \vee \overline{c})$$

$\vee$ ***is logic "or"***

$\wedge$ ***is logic "and" or "logical conjunction"***

Truth assignments: $a = true, b = true, c = \ false \Rightarrow$

$$\textbf{the entire expression} = true$$

Checking phase: Θ(*n*)

Deterministic VS Nondeterministic Algorithm

A problem can have BOTH *deterministic* and *nondeterministic* algorithms

Example:

Shortest path problem: find the shortest path from ***a*** to ***b*** in a weighted graph

- **Deterministic algorithm:** searching the shortest path (brute force enumerating)
- **Nondeterministic algorithm:** generate a path ***P*** and decide whether ***P*** is a simple path (all vertices on the path are distinct) from ***a*** to ***b*** of length<= Threshold.

The Class P & *NP*

P: the class of decision *problems* that are solvable by deterministic algorithms in $O(p(n))$, where $p(n)$ is a polynomial on ***n***

NP: the class of decision *problems* that are solvable in polynomial time by *nondeterministic* algorithms

Thus *NP* can also be thought of as the class of problems

- whose solutions can be verified in polynomial time; or
- that can be solved in polynomial time on a machine that can pursue infinitely many paths of the computation in parallel

Note that *NP* stands for "Nondeterministic Polynomial-time"

All the problems in *P* can also be solved in this manner (but no guessing is necessary), so we have:

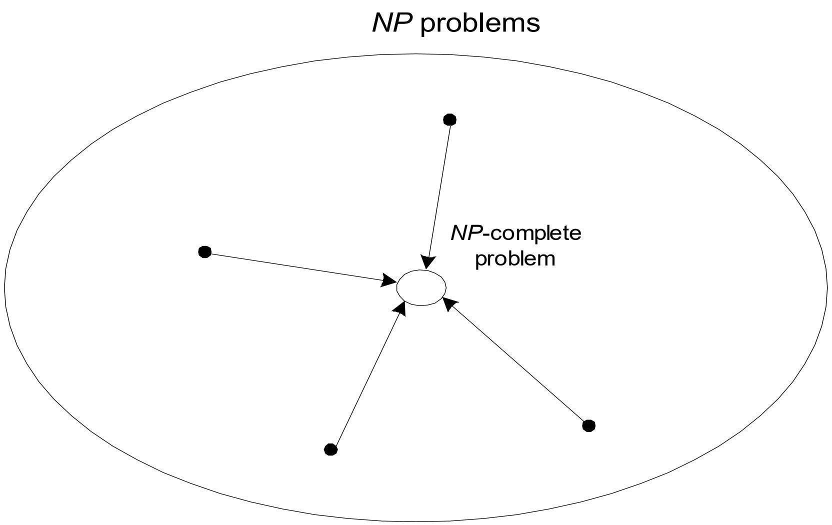
$$P \subseteq NP$$

NP-Complete problems

A decision problem *D* is <u>*NP*-complete</u> iff

1. $D \in NP$
2. every problem in *NP* is polynomial-time reducible to *D*

The class of *NP*-complete problems is denoted <u>*NPC*</u>



Polynomial Reductions

A decision problem D_1 is said to be polynomial reducible to a decision problem D_2 if there exists a function *f* that transforms instances of D_1 to instances of D_2 such that

1. *f* maps all "yes" instances of D_1 to "yes" instances of D_2 and all "no" instances of D_1 to "no" instances of D_2

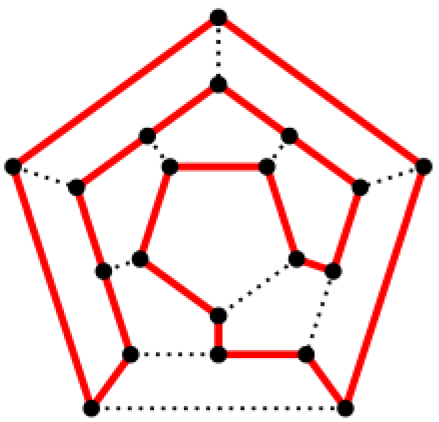
2. *f* is computable by a polynomial-time algorithm

If D_2 can be solved in polynomial time → D_1 can be solved in polynomial time

Polynomial Reductions

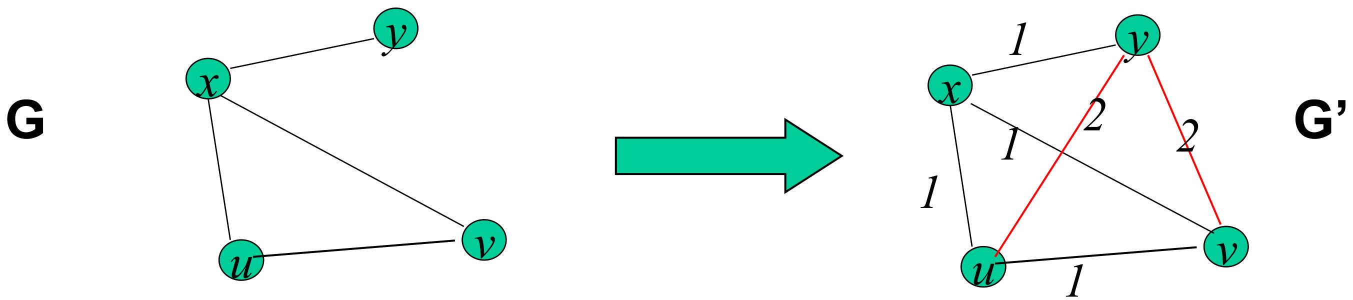
Example: Polynomial-time reduction of Hamiltonian Circuit to decision version of Traveling Salesman Problem (Is there a solution of TSP with total distance no larger than *k=n?*) given integer distance

Hamiltonian Circuit: a closed path in a graph that visits every node in the graph exactly once



Traveling Salesman: find the shortest path that visits every city exact once and returns to the origin

Polynomial Reductions



- If G has a Hamiltonian cycle, G' has a cycle w/ weight ***n***

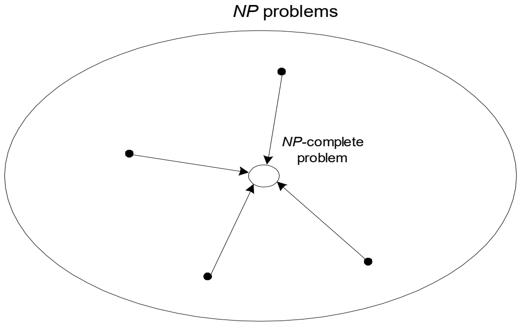
What does this prove?

- If HC is NPC → TSP(D) is NPC? or
- If TSP(D) is NPC → HC is NPC?

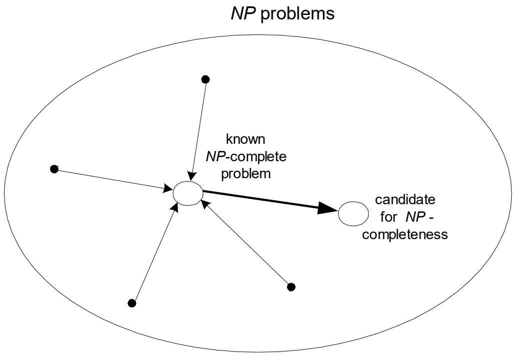
To Prove a Decision Problem is in NPC

1. Prove it is in *NP* (verification takes polynomial time)

2. Prove that all problems in *NP* is reducible to this problem



3. Or Prove that a known *NPC* problem is reducible to this problem



BIG problem: If we can prove any given NPC problem can be solve in polynomial time → P=NP