CS 6212 DESIGN AND ANALYSIS OF ALGORITHMS

LECTURE: NP-COMPLETE THEORY – PART I

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE

By the end of this lecture, you will be able to:

- Describe the definitions of NP, polynomial problems, exponential problems, intractable/undecidable problems
- Explain what yes-no problems are, and convert optimization problems into yes-no problems
- Develop NP algorithms for NP problems
- Describe in preliminary terms polynomial-time transforms between yes-no problems

OUTLINE

- Taxonomy of computational problems
- Yes-no problems
- The NP class/family and its significance
- Definition of NP problems and NP algorithms
- Development of NP algorithms for a number of NP problems

INTRODUCTION
-- POLYNOMIAL-TIME PROBLEMS --

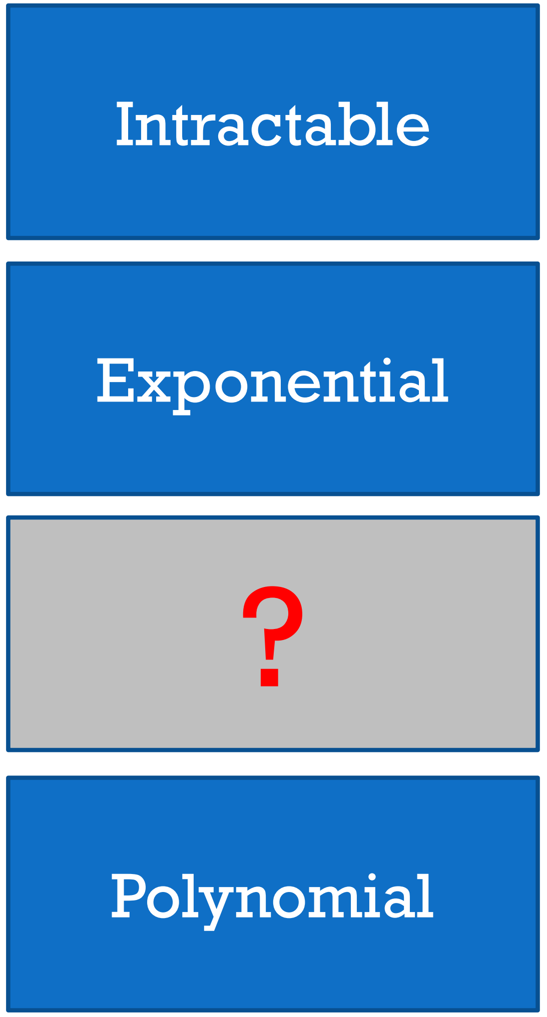
- **Definition**: A problem is said to be ***polynomial*** if there is an algorithm that solves the problem in time $T(n) = O(n^c)$, where c is a constant.
- Examples of polynomial problems:
  - Sorting: $O(n \log n) \ = \ O(n^2)$
  - All-pairs shortest path: $O(n^3)$
  - Minimum spanning tree: $O(E \log E) = \ O(E^2)$

INTRODUCTION
-- EXPONENTIAL-TIME PROBLEMS --

- **Definition**: A problem is ***exponential*** if no polynomial-time algorithm can be developed for it and if we can find an algorithm that solves it in $O(n^{u(n)})$, where $u(n) \to \infty$ as $n \to \infty$.
- Examples of exponential problems:
  - Generating combinatorial families of exponential size (like subsets of a set, binary strings, graphs, permutations, etc.)
  - The Tower of Hanoi problem

TAXONOMY OF COMPUTATIONAL PROBLEMS
-- FOUR BROAD CLASSES OF PROBLEMS --

- The world of computation can be subdivided into 3 classes:
  - Polynomial problems (P)
  - Exponential problems (E)
  - Intractable/undecidable (non-computable) problems (I)
- There is a very large and important class of problems that
  - we know how to solve exponentially,
  - we don't know how to solve polynomially, and
  - we don't know if they can be solved polynomially at all
- This class is a gray area between the P-class and the E-class. It will be studied in this and next lecture



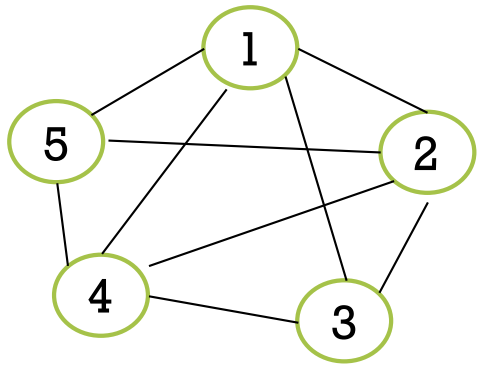
FOCUS ON YES-NO PROBLEMS

- **Definition**: A *yes-no problem* consists of an instance (or input I) and a yes-no question Q.
- Yes-no problems are also called *decision problems*
- Much of NP-complete theory focuses on yes-no problems
- Several examples of yes-no problems follow

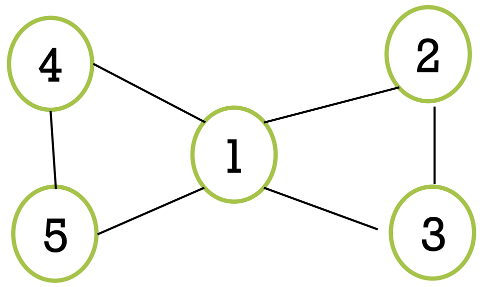
AN EXAMPLE OF YES-NO PROBLEMS
-- THE HAMILTONIAN CYCLE PROBLEM (HAM) --

The Hamiltonian Cycle (HC) problem HAM is:

- **Input**: A graph G
- **Question**: Does G have a Hamiltonian Cycle?



This graph does

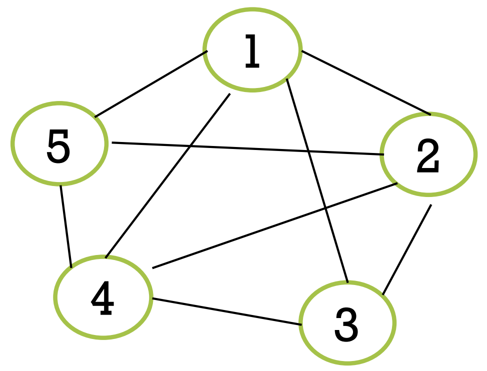


This graph does not

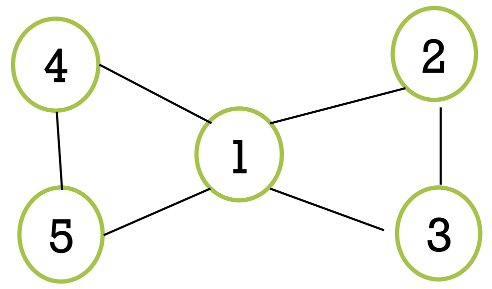
ANOTHER EXAMPLE OF YES-NO PROBLEMS -- THE CLIQUE PROBLEM--

The K-clique problem (CLIQUE):

- **Input**: A graph G and an integer k
- **Question**: Does G have a k-clique?



This graph has 4-cliques



This graph does not have 4-clqiues

ANOTHER EXAMPLE OF YES-NO PROBLEMS
-- THE SUBSET-SUM PROBLEM --

- The subset-sum problem

  Instance: A real array A[1:n]

  Question: Can A be partitioned to 2 parts of equal sum?

- Examples:

  1. A=[1, 2, 4, 2, 7]. The answer is YES:

     Partition A to **{1,7}** and **{2,4,2}**, both adding up to 8.

  2. A=[2,4,2,7]. The answer to the question is NO. Why?

ANOTHER EXAMPLE OF YES-NO PROBLEMS
-- THE SATISFIABILITY PROBLEM (SAT) --

- The satisfiability problem (SAT)

  Instance: A Boolean Expression F

  Question: Is there an assignment to the variables in F so that F evaluates to 1?

Boolean algebra:
- A Boolean variable x can be assigned 0 or 1
- There are 3 Boolean operations: $+ \quad . \quad '$ standing for "or", "and", "not"
- 0+0=0, 0+1=1+0=1, 1+1=1
- 0.0=0, 0.1=1.0=0, and 1.1=1
- $0' = 1, 1' = 0$ (note that $x.x' = 0\ \forall$ Bool x)
- A Boolean expression is an expression involving Boolean variables and the three operations

$xy = x.y$

- Examples:

1. $F = (x_1 + x_2')(x_1' + x_2 + x_3)$. Answer is YES:

   $x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0;\ F = (1+1')(1'+1+0) = (1+0)(0+1+0) = 1\,.\,1 = 1$

2. $F = (x_1 + x_2)x_1'x_2'$. Answer is NO. (Why?)

   Try every possible 0/1 assignment to x_1 and x_2, and evaluate F. You will find that F=0 in each case

ANOTHER EXAMPLE OF YES-NO PROBLEMS
-- THE TRAVELING SALESMAN PROBLEM (TSP) --

- The original formulation of the Traveling Salesman Problem

  Instance: A weighted graph G

  Question: Find a minimum-weight Hamiltonian cycle in G.

- The **yes-no** formulation of TSP:

  Instance: A weighted graph G and a real number d

  Question: Does G have a Hamiltonian cycle of weight $\leq$ d?

In the context of TSP, a HC is called a *traveling salesman tour*

DEFINITION OF NP
-- USING TURING MACHINES --

- **Definition 1 of NP**: A yes-no problem is said to be *Non-deterministically Polynomial* (***NP***) if we can find a non-deterministic Turing machine that can solve the problem in a polynomial number of nondeterministic moves.

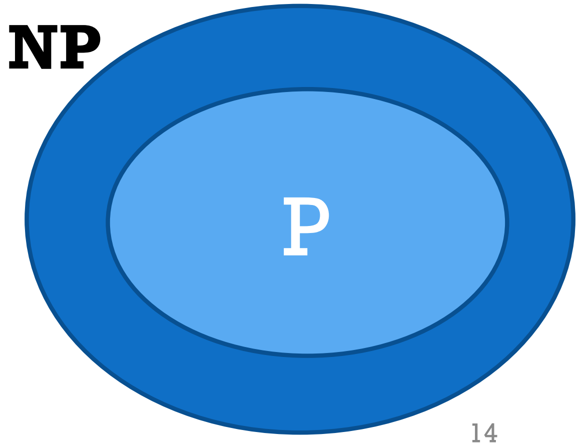
- For those who are not familiar with Turing machines, two alternative definitions of NP will be developed.

DEFINITION OF NP

-- ALTERNATIVE DEFINITION: THE GRADING ANALOGY --

- **Definition 2 of NP**: A yes-no problem is said to be NP if
  - its solution comes from a finite set of possibilities, and
  - it takes polynomial time to verify the correctness of a candidate solution
- **Remarks**
  - It is much easier and faster to "grade" (i.e., verify) a solution than to find a solution from scratch.
  - We use NP to designate the class of all non-deterministically polynomial problems.
  - Clearly, P is a subset of NP (Why?)
- A very famous open question in Computer Science:

$$P \overset{?}{=} NP$$



DEFINITION OF NP ALGORITHM

-- THE IMAGINARY "CHOOSE" -

- To give the 3rd alternative definition of NP, we introduce an imaginary, non-implementable instruction, which we call "**choose()**".
- Behavior of "choose()": genius-crazy
  - if a problem has a solution of N components, choose(i) magically returns the i-th component of the CORRECT solution in constant time
  - if a problem has no solution, choose(i) returns mere "garbage", that is, it returns an uncertain value (instead of knowing and saying "there is no solution")
- **Definition of NP algorithm**: An NP algorithm is an algorithm that has 2 stages:
  1. **A guessing stage** that uses choose() to find a solution to the problem
  2. **A verification stage** that checks in **polynomial time** the correctness of the solution produced by the first stage, **without using choose()**

TEMPLATE OF NP ALGORITHMS

```
begin
    // the guessing stage: guesses solution X[1:N] in O(N) time
    for i=1 to N do
        X[i] := choose(i);
    endfor

    // the verification stage
    Write code that does not use "choose" and that verifies in
    polynomial time if X[1:N] is a correct solution to the
    problem. If correct, return (yes); else, return no.
end
```

The guessing stage (uses "choose")

The verification stage (NO "choose")

DEFINITION OF NP

-- ALTERNATIVE DEFINITION: THE IMAGINARY "CHOOSE" --

- **Definition 3 of NP**: A yes-no problem is said to be NP if there exists an NP algorithm for it (i.e., a polynomial guess-verify algorithm).
- **Remark**: For the NP algorithm template to be polynomial, the solution size N must be polynomial in the input size n, and the verification stage must be polynomial in n.
- **Note**: We will use the third definition of NP

AN NP ALGORITHM FOR HAM

```
Function HC( input: G)
begin
    // the guessing stage: guesses solution X[1:n] in O(n) time
    for i=1 to n do
        X[i] := choose(i);
    endfor

    // the verification stage.  Time: $O(n^2)$, which is polynomial
    for i=1 to n do
        for j=i+1 to n do
            if (X[i] == X[j]) then                    return(no);      endif
        endfor
    endfor
    for i=1 to n-1 do
        if ((X[i],X[i+1]) is not an edge) then        return(no);      endif
    endfor
    if ((X[n],X[1]) is not an edge) then              return(no);      endif
    return(yes);
end
```

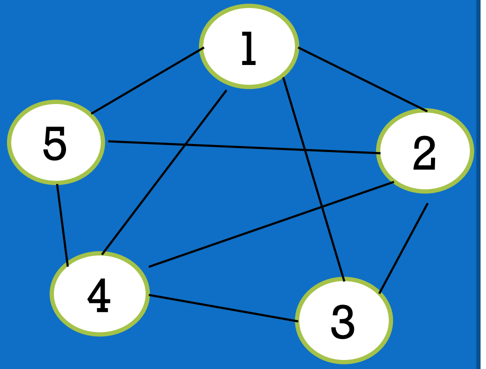
- The HC algorithm is $O(n^2)$, which is polynomial
- Therefore, the HC problem is NP

AN NP ALGORITHM FOR THE CLIQUE PROBLEM

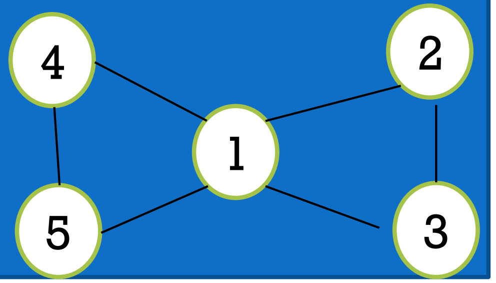
```
Function kclique( input: G, k)
begin
    // the guessing stage: guesses solution
    X[1:k] in O(n) time
    for i=1 to k do
        X[i] := choose(i);
    endfor

    // the verification stage.
    for i=1 to k do
        for j=i+1 to k do
            if (X[i] == X[j] || (X[i],X[j]) is not
            an edge) then
                return(no);
            endif
        endfor
    endfor
    return(yes);
end
```

- The HC algorithm is $O(k^2) = O(n^2)$, which is polynomial
- Therefore, the HC problem is NP

This graph has 4-cliques:



This graph does not have 4-clqiues:



The K-clique problem

- **Input**: A graph G and an integer k
- **Question**: Does G have a k-clique?

CONNECTION B/W THE VERIFY STAGE AND THE BOUND FUNCTION IN BACKTRACKING

- Recall that in Backtracking, the Bound function checked for validity of a (partial) solution.
    - It did the checking incrementally, i.e., on a new component X[r]
    - Assuming X[1:r-1] is all valid
- The verification stage in NP algorithms is like the Bound function, except that the former checks the validity of whole solution all at once
- Despite the difference, the underlying logic is the same

LESSONS LEARNED SO FAR

- There are some problems, called NP problems, that can be solved in exponential time, but we don't know if they are inherently exponential or can one day be solved in polynomial time
- Proving a yes-no problem to be NP is easy: design a guess-verify algorithm (where the guess stage uses "choose" but the verify stage does not), such that the verify stage takes polynomial time
- The verify-stage is similar to the Bound function in Backtracking

EXERCISES

- **Exercise 1**: Give an NP algorithm for the SUBSET-SUM problem
- **Exercise 2**: Give an NP algorithm for the TSP
- **Exercise 3**: Give an NP algorithm for the SAT problem
- **Exercise 4**: An independent set of k nodes in a graph G is any subset of k nodes in G such that no two nodes are adjacent. Express as a yes-no problem the problem of whether G has a k-node independent set, and prove it to be NP by giving an NP algorithm for it.
- **Exercise 5**: A vertex cover C of k nodes in a graph G is any subset of k nodes in G such that every edge in G has at least one end node in C. Express as a yes-no problem the problem of whether G has a k-node vertex cover, and prove it to be NP.
- **Exercise 6**: Express the k-colorability of a graph G as a yes-no problem, and prove it to be NP.

NEXT LECTURE

- Transforms for yes-no problems
- Generalizations of transforms and some applications
- Reductions (using transforms)
- Definition of NP-completeness, conceptually first, and using reductions second
- Practical strategy for proving new problems to be NP-complete
- Proving CLIQUE to be NP-complete, using a transform/reduction
- Closing thoughts