Computational Models - Lecture 11[1]

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

May 18/20, 2014

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Reminder – deterministic and nondeterministic time classes
- Additional NP languages
- The class *co-*$\mathcal{NP}$
- $\mathcal{P}$ Verses $\mathcal{NP}$
- NP-completeness
- Satisfiability
- Reductions

- Sipser's book, 7.4–7.5

Reminder – Deterministic Time

Definition 1 (deterministic Time)

A deterministic TM M runs in time t, where $t : \mathbb{N} \mapsto \mathbb{N}$, if for every input w, the number of steps that M(w) uses is at most $t(|w|)$.

Definition 2 (DTIME)

For $t : \mathbb{N} \mapsto \mathbb{N}$, let
DTIME(t) $= \{L \subseteq \Sigma^* :$ L is decided by an $O(t)$-time single tape TM$\}$

The bound on the running time is required to hold also for input not in the language (i.e., L).

Definition 3 ($\mathcal{P}$)

$\mathcal{P} = \bigcup_{c \geq 0}$ DTIME(n^c)

Reminder – Non-Deterministic Time

Definition 4 (nondeterministic Time)

A non-deterministic TM N runs in time t, where $t: \mathbb{N} \mapsto \mathbb{N}$, if for every input w, the maximum number of steps that $N(w)$ uses on any branch of its computation tree, is at most $t(|w|)$.

Notice that also non-accepting branches must reject within the required time.

Definition 5 (NTIME)

For $t \mapsto \mathbb{N} \mapsto \mathbb{N}$ let $\text{NTIME}(t) =$
$\{L \subseteq \Sigma^* : \ L$ is decided by an $O(t)$-time single tape NTM$\}$

Definition 6 ($\mathcal{NP}$)

$\mathcal{NP} = \bigcup_{c \geq 0} \text{NTIME}(n^c)$

Reminder – Non-Deterministic Time, cont.

Definition 7 (verifier)

Algorithm V is a verifier for a language L, if

- $x \in L \implies \exists c \in \Sigma^*$ s.t. $V(x,c) = 1$.
- $x \notin L \implies \nexists c \in \Sigma^*$ s.t. $V(x,c) = 1$.

A polynomial verifier runs in polynomial time in $|x|$ (i.e., in the length of its left-hand-side input parameter).

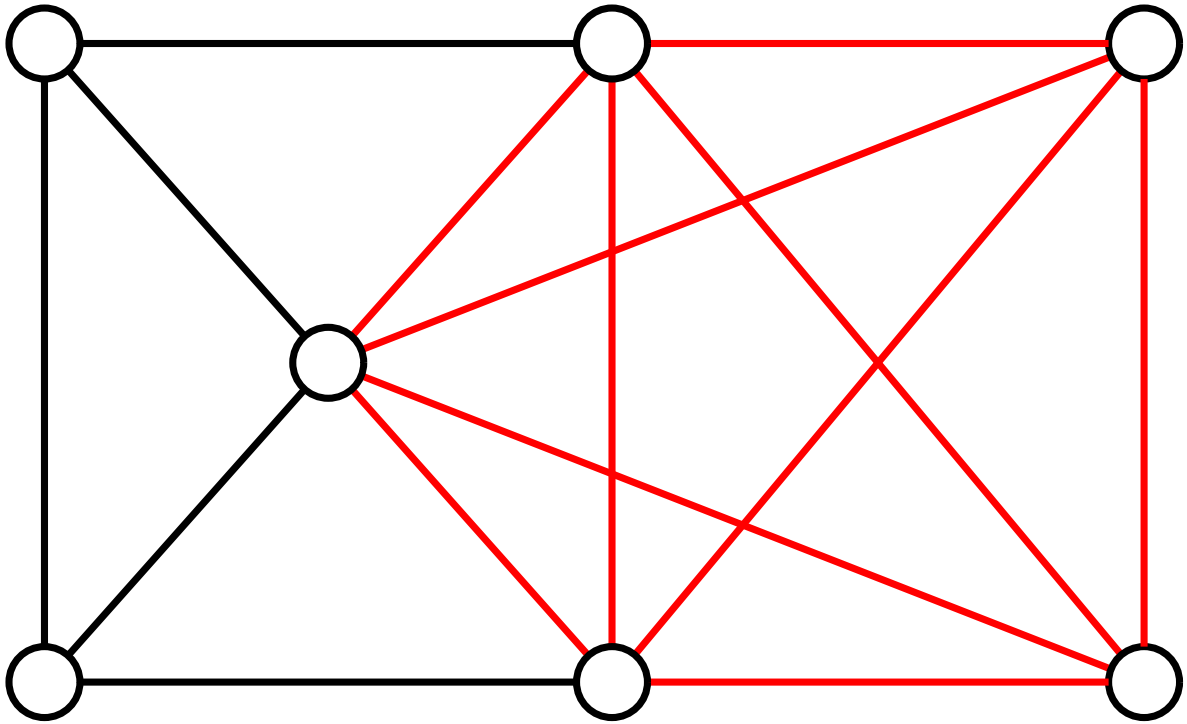
Theorem 8

A language is in $\mathcal{NP}$ iff it has a polynomial time verifier.

Section 1

Additional NP Languages

CLIQUE



A clique in a graph is a subgraph where every two nodes are connected by an edge.

A k-clique is a clique of size k.

Question 9

What is the largest k-clique in the figure?

CLIQUE cont.

$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\}$$

Theorem 10

CLIQUE $\in \mathcal{NP}$

Proof's idea: The clique is the certificate.

Algorithm 11 (V)

On input $(\langle G, k\rangle, c)$
`Accept` if c is a k-clique subgraph of G;
Otherwise `Reject`.

SUBSET-SUM

- A collection of non-negative integers $x_1, \ldots, x_k$
- A target number t

Question 12

does some sub-collection add up to t?

Example 13

- $\langle \{4, 11, 16, 21, 27\}, 25 \rangle \in$ SUBSET-SUM because $4 + 21 = 25$.
- $\langle \{4, 11, 16, 21, 27\}, 26 \rangle \notin$ SUBSET-SUM (?)

SUBSET-SUM cont.

$$\text{SUBSET-SUM} = \{\langle S = \{x_1, \ldots, x_k\}, t\rangle : \exists \{y_1, \ldots, y_\ell\} \subseteq S: \ \sum_{j=1}^{\ell} y_j = t\}$$

Collections are sets: repetitions not allowed.

Theorem 14

SUBSET-SUM $\in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 15 (V)

On input $(\langle S = \{x_1, \ldots, x_k\}, t\rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. c is a collection of numbers summing to t.
2. c is a subset of S

Theorem 16

There is a pseudo-polynomial algorithm for SUBSET-SUM

Graph characterization of SUBSET-SUM

Definition 17

For $\langle S = \{x_1, \ldots, x_k\}, t\rangle$, let $G(S, t) = (V, E)$

- $V = V_0 \bigcup V_1 \ldots \bigcup V_k$, where $V_i = \{v_{i,0}, \ldots, v_{i,t}\}$
- $E = E_1 \bigcup \ldots \bigcup E_k$, where $E_i = \{(v_{i-1,j}, v_{i,j}), (v_{i-1,j}, v_{i,j+x_i}) : j \in \{0, \ldots, t\}\}$

Claim 18

$v_{i,j}$ is reachable from $v_{0,0}$ in $G(S, t)$, iff $\langle\{x_1, \ldots, x_i\}, j\rangle \in$ SUBSET-SUM.

Proof?

Pseudo-polynomial algorithm for SUBSET-SUM

Algorithm 19

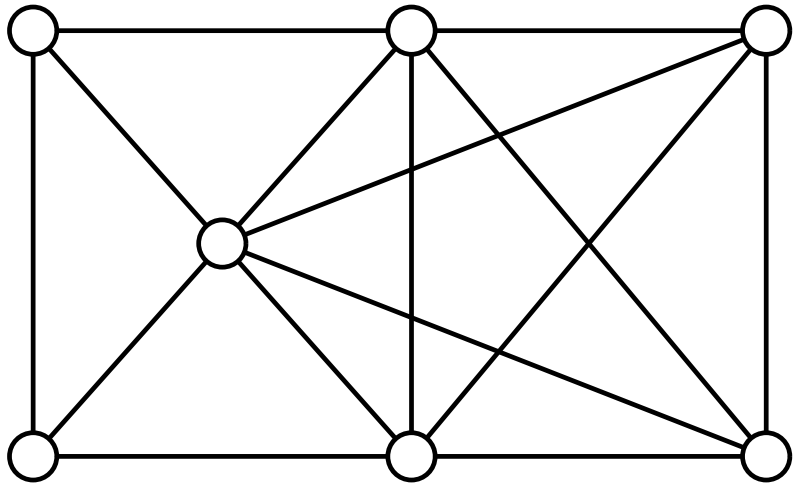
Input: $\langle S, t \rangle$.

Return TRUE iff $v_{k,t}$ is reachable from $v_{0,0}$ in $G(S, t)$.

- Correctness
- Size of $G(S, t)$ is $O(|S| \cdot t)$, and therefore the resulting algorithm for SUBSET-SUM runs in time $\text{poly}(|S| \cdot t)$.

  Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Independent Set



An independent set in a graph is a set of vertexes, no two of which are linked by an edge.

A k-IS is an independent set of size k.

Question 20

What is the largest k-IS in the figure?

IND-SET cont.

$$\text{IND-SET} = \{\langle G, k\rangle : G \text{ contains an independent set of size } k\}$$

Theorem 21

IND-SET $\in \mathcal{NP}$

Proof's idea: The independent set is the certificate.

Algorithm 22 (V)

On input $(\langle G, k\rangle, c)$
`Accept` if c is a k-IS of G (no edges between nodes in c, and $|c| = k$);
Otherwise `Reject`.

KNAPSACK

- A collection of non-negative integers $x_1, \ldots, x_k$ (size)
- A collection of non-negative integers $y_1, \ldots, y_k$ (benefit)
- A capacity B
- A target number t

Question: does some $S \subseteq \{1, \ldots, k\}$ has $\sum_{i \in S} x_i \leq B$ and $\sum_{i \in S} y_i \geq t$?

Example 23

- $(\{4, 11, 16, 21, 27\}, \{7, 3, 9, 5, 35\}, B = 20, t = 16) \in KNAPSACK$ because $S = \{1, 3\}$.

KNAPSACK cont.

$$\text{KNAPSACK} = \{\langle\{x_1,\ldots,x_k\},\{y_1,\ldots,y_k\},B,t\rangle : \exists S \subseteq \{1,\ldots,k\} : \sum_{i\in S} x_i \leq B \wedge \sum_{i\in S} y_i \geq t\}$$

Theorem 24

KNAPSACK $\in \mathcal{NP}$

Proof's idea: The subset is the certificate.

Algorithm 25 (V)

On input $(\langle\{x_1,\ldots,x_k\},\{y_1,\ldots,y_k\},B,t\rangle, c)$:
`Accept` if the following holds (otherwise `Reject`):

1. $\sum_{i\in c} x_i \leq B$
2. $\sum_{i\in c} y_i \geq t$.
3. c is a subset of $[1,\ldots,k]$

Pseudo-polynomial algorithm for KNAPSACK

Theorem 26

There is a pseudo-polynomial algorithm for KNAPSACK

Algorithm 27 (KNAPSACK)

Input: $\{x_1, \ldots, x_k\}, \{y_1, \ldots, y_k\}, B, t$:

1. Set $A(0,0) = 0$ and $A(0,p) = \infty$ for $p \neq 0$.
2. For $i = 1$ to k:

   For $p = 0$ to t:

   Set $A(i,p) = \min\{A(i-1,p), x_i + A(i-1,p-y_i)\}$.
3. `Accept` if $t \leq \max\{p\colon A(k,p) \leq B\}$.

- Running time $O(k \cdot t)$
- Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Section 2

The Class coNP

The Class *co-*$\mathcal{NP}$

$\overline{\text{CLIQUE}} = \{\langle G, k\rangle : G$ is an undirected graph with no k-clique$\}$ seems not to be member of $\mathcal{NP}$.

It is harder to efficiently verify that something does not exist than to efficiently verify that something does exist.

Definition 28 (*co-*$\mathcal{NP}$)

co-$\mathcal{NP} = \{\mathsf{L} : \overline{\mathsf{L}} \in \mathcal{NP}\}$.

So far, no one knows if *co-*$\mathcal{NP}$ is distinct from $\mathcal{NP}$.

Claim 29

$\mathcal{P} \subseteq$ *co-*$\mathcal{NP}$.

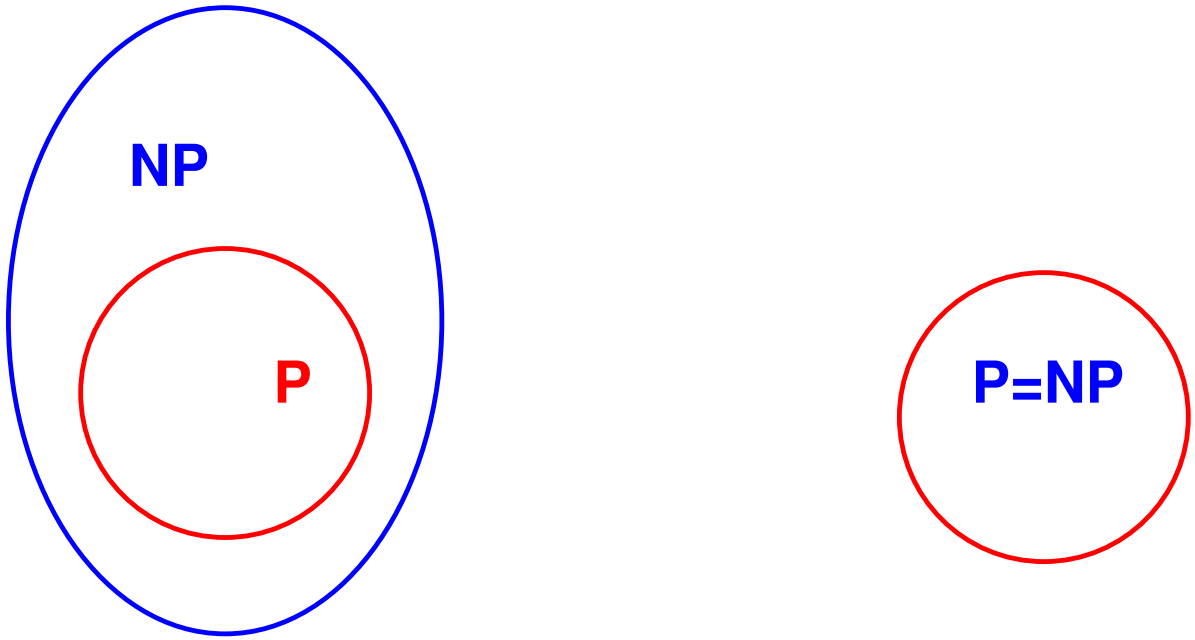
Proof?

$$\mathsf{L} \in \mathcal{P} \implies \overline{\mathsf{L}} \in \mathcal{P} \implies \overline{\mathsf{L}} \in \mathcal{NP} \implies \mathsf{L} \in \textit{co-}\mathcal{NP}$$

Section 3

P vs. NP

$\mathcal{P}$ Vs. $\mathcal{NP}$



The question $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ is one of the great unsolved mysteries in contemporary mathematics.

- Most computer scientists believe the two classes are not equal
- Most bogus proofs show them equal (?)

$\mathcal{P}$ Vs. $\mathcal{NP}$, cont.

If $\mathcal{P}$ differs from $\mathcal{NP}$, then the distinction between $\mathcal{P}$ and $\mathcal{NP} \setminus \mathcal{P}$ is meaningful and important.

- languages in $\mathcal{P}$ are tractable
- languages in $\mathcal{NP} \setminus \mathcal{P}$ are intractable

Until we can prove that $\mathcal{P} \neq \mathcal{NP}$, there is no hope of proving that a specific language lies in $\mathcal{NP} \setminus \mathcal{P}$.

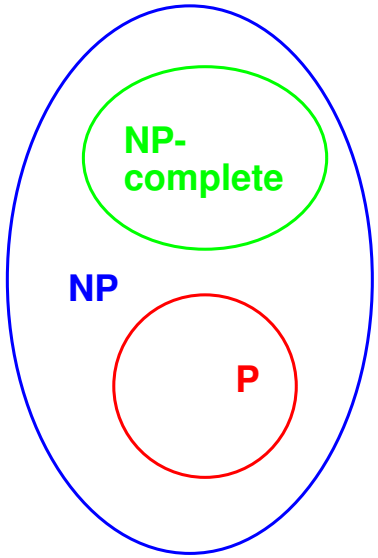
Nevertheless, we can prove statements of the form

If $\mathsf{A} \in \mathcal{NP} \setminus \mathcal{P}$, *then* $\mathsf{B} \in \mathcal{NP} \setminus \mathcal{P}$.

Section 4

NP Completeness

NP Completeness



The class of NP-complete languages are

- "hardest" languages in $\mathcal{NP}$
- "least likely" to be in $\mathcal{P}$
- If any NP-complete L $\in P$, then $\mathcal{NP} = \mathcal{P}$.

Such languages, "carry on their backs" the burden of all of $\mathcal{NP}$.

Question 30

Are there NP-complete languages?

Polynomial-Time Computable Functions

Definition 31 (poly-time computable functions)

A function $f : \Sigma^* \mapsto \Sigma^*$ is polynomial-time computable, if there is a poly-time deterministic TM that

- starts with input w, and
- halts with $f(w)$ on tape.

Polynomial-Time Reducibility

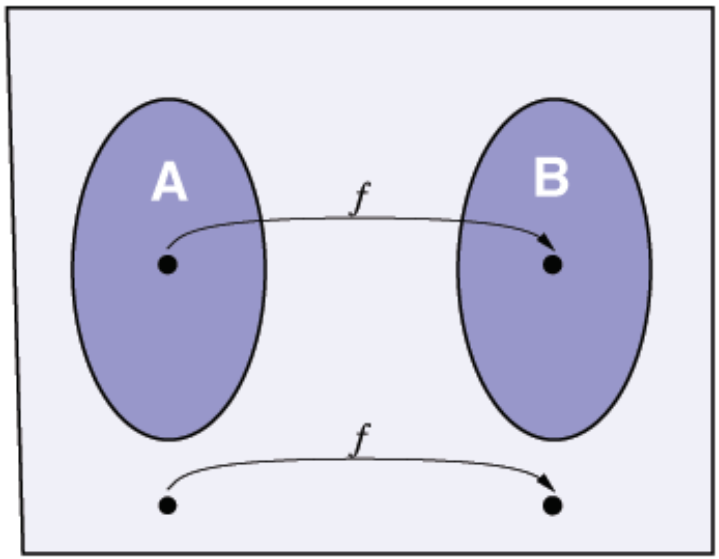
Definition 32

A language A is polynomial time mapping reducible to B, denoted $A \leq_P B$, if exists poly-time computable f such that

$$w \in A \iff f(w) \in B.$$

for every $w \in \Sigma^*$.
The function f is called a polynomial-time reduction from A to B.

The mapping f efficiently converts questions about membership in A to membership in B.



Reductions to $\mathcal{P}$

Theorem 33

If $A \leq_P B$ *and* $B \in \mathcal{P}$ *then* $A \in \mathcal{P}$.

Proof:

- Let f the reduction from A to B, computed by TM M_f.
  On input x, the TM M_f makes at most $c_f \cdot |x|^{a_f}$ steps.
- Let M_B be the poly-time decider for B.
  On input y, the TM M_B makes at most $c_B \cdot |y|^{a_B}$ steps.

Algorithm 34 (Decider M_A for A)

On input x, return $M_B(f(x))$

- M_A decides A
- Running time of $M_B(x)$, is at most
  $c_B \cdot (c_f \cdot |x|^{a_f})^{a_B} = (c_B \cdot c_f^{a_B}) \cdot |x|^{a_f \cdot a_B} \in \text{poly}(|x|)$
  Hence, $A \in \mathcal{P}$

What $A \leq_P B$ tells us about B?

Question 35

Assume that $\{0^n1^n : n \geq 0\} \leq_P L$. Does it yield that $L \in \mathcal{P}$?

Answer: No.

Let $L = H_{TM}$ and define $f(x) = \begin{cases} M_{stop}, & x \in \{0^n1^n : n \in \mathbb{N}\} \\ M_{no-stop}, & \text{otherwise.} \end{cases}$.

$A \leq_P B$ does tell us that B is “at least as hard" as A.

NP Completeness, Formal Definition

Definition 36 ($\mathcal{NP}$-complete)

A language B is NP-complete, if

- B $\in \mathcal{NP}$, and
- Every A $\in \mathcal{NP}$ is poly-time reducible to B (i.e., A $\leq_P$ B)

We let $\mathcal{NPC}$ denote the class of all NP-complete languages.

Compare to

Definition 37 (RE-Complete)

A language B is RE-complete, if

- B $\in \mathcal{RE}$, and
- Every A $\in \mathcal{RE}$ is mapping reducible to B.

Why NP Completeness?

Theorem 38

If $\mathsf{B} \in \mathcal{NPC}$ *and* $\mathsf{B} \in \mathcal{P}$, *then* $\mathcal{P} = \mathcal{NP}$.

Proof: Immediately follows by Thm 33. ♣

To show $\mathcal{P} = \mathcal{NP}$ (and make an instant fortune, see www.claymath.org/millennium/P_vs_NP/), suffices to find a polynomial-time algorithm for any NP-complete problem.

Question 39

Is $\mathcal{NPC}$ empty?

$\mathcal{NPC}$ Is Not Empty

$A_{NP} = \{\langle M, x, 1^n\rangle : M \text{ is a TM} \wedge \exists c \in \Sigma^* \text{ s.t. } M(x, c) \text{ accepts within } n \text{ steps}\}$.

Theorem 40

$A_{NP} \in \mathcal{NPC}$

Proof:

- Clearly $A_{NP} \in \mathcal{NP}$.(?)
- Let $L \in \mathcal{NP}$, let V be a verifier for L and let $p \in poly$ be a bound on the running time of V (i.e., $V(x, \cdot)$ halts within $p(|x|)$ steps, for every $x \in \Sigma^*$)).
- Define $f(x) = \langle V, x, 1^{p(|x|)}\rangle$.
- Clearly f is poly-time computable and $x \in L \iff f(x) \in A_{NP}$.

♣

Finding Additional NP-complete Languages

Theorem 41

Assume that

1. $B \in \mathcal{NP}$
2. $A \in \mathcal{NPC}$ *and* $A \leq_P B$

then $B \in \mathcal{NPC}$.

Proof: Home exercise . . . ♣

We would like to find $L \in \mathcal{NPC}$ that is “natural" and “easy" to reduce to.

The most useful such language is the language of satisfied formulas.

Section 5

Satisfiability

Boolean Variables

- A Boolean variable assumes values
  - TRUE (written 1), and FALSE (written 0).
- Boolean operations:
  - and: $\wedge$
  - or: $\vee$
  - not: $\neg$
- Examples:

$$\begin{aligned} 0 \wedge 1 &= 0 \\ 0 \vee 1 &= 1 \\ \overline{0} &= 1 \end{aligned}$$

Boolean Formulas and SAT

A Boolean formula is an expression involving Boolean variables and operations.

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Definition 42 (satisfiable formula)

A formula is satisfiable, if some Boolean assignment to its variables, makes the formula evaluate to 1.

The formula $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ is satisfiable by the assignment

$$\begin{aligned} x &= 0 \\ y &= 1 \\ z &= 0 \end{aligned}$$

The language of satisfied formulas:

$$\text{SAT} = \{\langle\phi\rangle : \ \phi \text{ is a satisfiable Boolean formula}\}$$

SAT $\in \mathcal{NPC}$

$$\text{SAT} = \{\langle \phi \rangle : \ \phi \text{ is satisfiable Boolean formula}\}$$

Theorem 43 (Cook-Levin (early 70s))

SAT $\in \mathcal{NPC}$.

- The “most important" $\mathcal{NP}$-complete language.
- It is easy to see that SAT $\in \mathcal{NP}$
- For the proof of other part wait for next week . . .

The Language 3SAT

It is useful to consider a special version of SAT

- A literal is a variable or negated variable: x or $\overline{x}$.
- A clause is several literals joined by $\vee$s: $(x_1 \vee \overline{x_2} \vee \overline{x_3})$
- A Boolean formula is in conjunctive normal form (CNF) if it consists of clauses, connected with $\wedge$s.

  For example: $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$

Definition 44

A Boolean formula is in k-CNF form, if it is a CNF formula, and all clauses have k literals.

- Example of 3CNF: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

The language of satisfied 3CNF formulas:

$$3SAT = \{\langle \phi \rangle : \ \phi \text{ is satisfiable 3CNF formula}\}$$

3SAT $\in \mathcal{NPC}$

- Clearly 3SAT $\leq_P$ SAT.

  and 3SAT $\in \mathcal{NP}$.
- We show next class that SAT $\leq_P$ 3SAT, yielding that 3SAT $\in \mathcal{NPC}$.
- Since 3SAT has more structure than SAT, it is simpler to reduce it to other languages.
- In the following we prove that several $\mathcal{NP}$ languages are $\mathcal{NPC}$, by showing a reduction from 3SAT to them.

1SAT, 2SAT $\in \mathcal{P}$

Question 45

Is 1SAT $\in \mathcal{P}$?

Question 46

Is 2SAT $\in \mathcal{P}$?

A Graph characterization for $2CNF$

Definition 47

For $2CNF$ formula ϕ with variables $x_1, \ldots, x_k$, define a directed graph $G(\phi) = (V, E)$ as

- $V = \{x_1, \overline{x_1}, \ldots, x_k, \overline{x_k}\}$
- $E = \{(\overline{\ell}, h), (\overline{h}, \ell) : (\ell \vee h) \in \phi\}$ (taking $\overline{\overline{x}} = x$).

Claim 48

$G(\phi)$ contains path from ℓ to $h \implies$ in any satisfying assignment of ϕ with $\ell = 1$, it holds that $h = 1$.

Proof?

Claim 49

$G(\phi)$ contains path from ℓ to $h \implies G(\phi)$ contains path from $\overline{h}$ to $\overline{\ell}$

Proof?

A Graph characterization for $2CNF$, cont

Definition 50

A variable x in $2CNF$ ϕ is bad, if $\exists$ paths in $G(\phi)$ from x to $\overline{x}$ and from $\overline{x}$ to x.

Claim 51

A $2CNF$ formula is satisfiable iff it contains no bad variables.

- ϕ has bad variable $\implies$ ϕ is unsatisfiable. Proof? by Claim 48
- ϕ has no bad variables $\implies$ ϕ is satisfiable. Proof?

Algorithm 52

While ϕ has unassigned variables:

1. Pick unassigned literal $\ell \in V$ that has no path from ℓ to $\overline{\ell}$
2. Assign 1 to all literals reachable from ℓ, and 0 to their negations.

Can there be conflicts (i.e., x and $\overline{x}$ assigned the same value)?

- Same iteration? Assume $\ell \rightsquigarrow h \rightsquigarrow \overline{h}$. $\implies$ $h \rightsquigarrow \overline{h} \rightsquigarrow \overline{\ell}$ $\implies$ $\ell \rightsquigarrow \overline{\ell}$.
- Different iterations? Removing edges cannot add bad variables.

Final assignment satisfies ϕ. Assume h is set to 1, then all clauses it participants in are satisfied.

Poly-time algorithm for 2SAT

Algorithm 53 (TwoSatSolver)

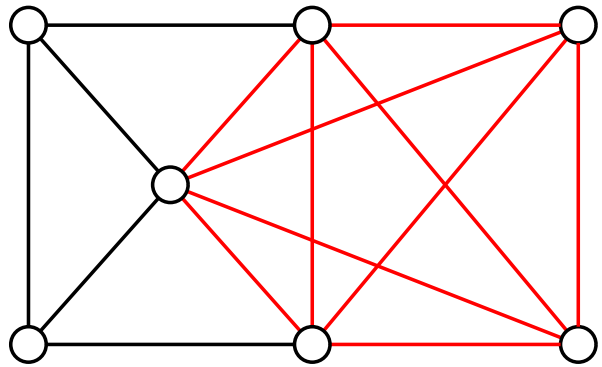
Input: *2CNF* ϕ
Return TRUE if there exist no bad variables in ϕ:

- Efficiency?
- Correctness?

Section 6

Clique

3SAT $\leq_P$ CLIQUE



$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\}$$

Claim 54

3SAT $\leq_P$ CLIQUE.

Since CLIQUE $\in \mathcal{NP}$, it follows that CLIQUE $\in \mathcal{NPC}$.

Proof's idea: We'll construct a poly-time reduction f that maps 3CNF formulae ϕ to pairs $\langle G, k\rangle$ of graphs and numbers.

The function f will have the property that ϕ is satisfiable, iff G has a clique of size k.

Proving 3SAT $\leq_P$ CLIQUE

On input ϕ, the mapping reduction is defined as follows:

If ϕ is not a 3CNF, output some $x \notin$ CLIQUE.

Otherwise, let k be the number of clauses in ϕ.

We construct a graph $G = G(\phi)$, see below, and output (G, k).

The graph G is defined as follows:

- Nodes in G are organized into triples $t_1, \ldots, t_k$.
- Each triple corresponds to a clause of ϕ
- Each node in a triple corresponds to a literal in corresponding clause.

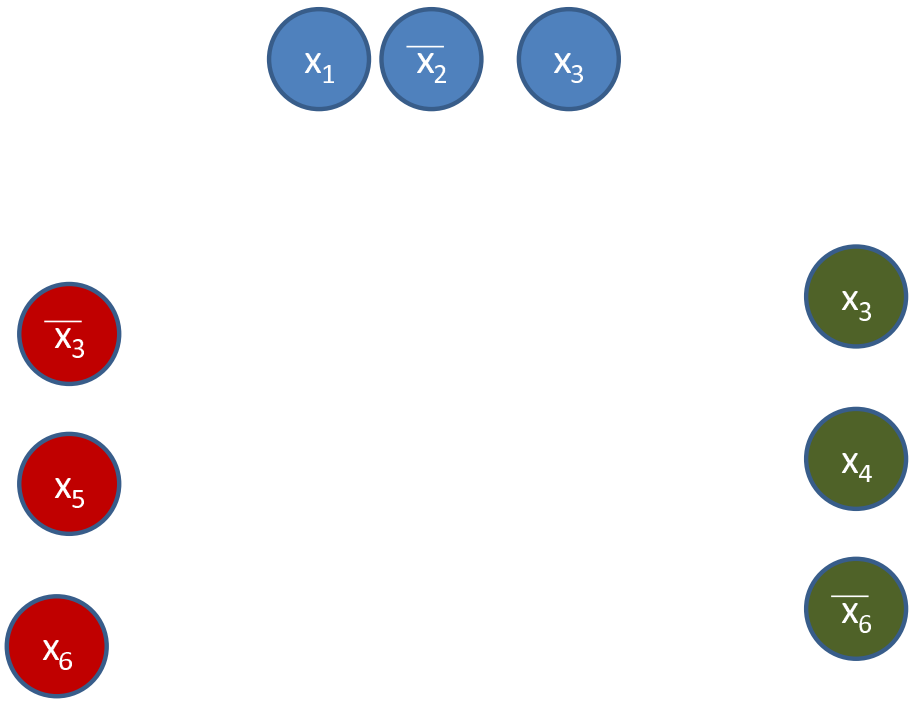
On going example:

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

Nodes of G

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$

Add a node per literal

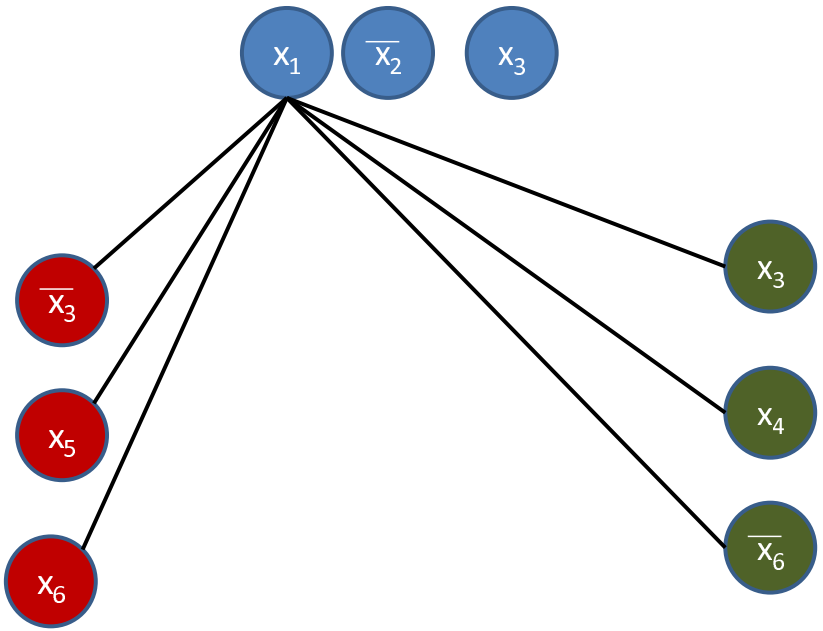


Edges of G

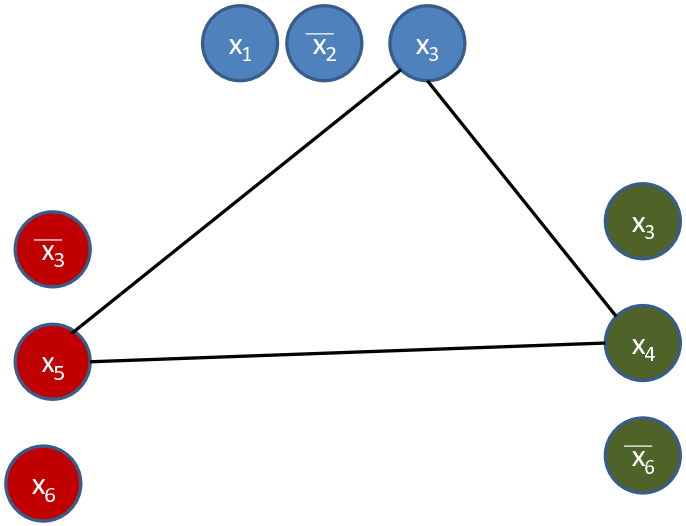
Add edges between all vertex pairs, except

- within same triple
- between contradictory literals (e.g., x_3 and $\overline{x_3}$)

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



$\phi \in 3SAT \implies \langle G, k \rangle \in CLIQUE$

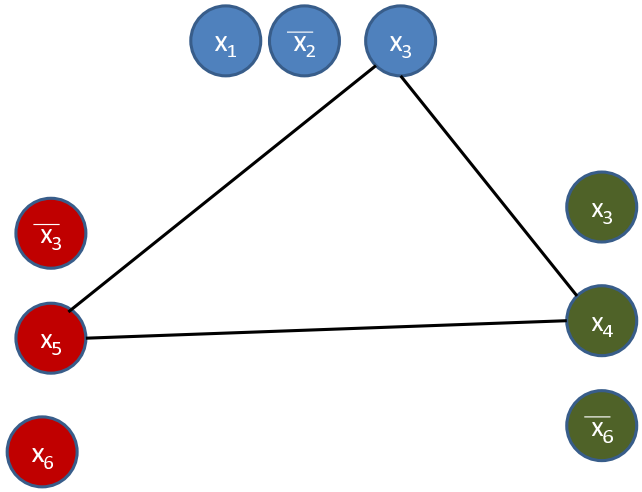


Proof: Suppose ϕ is satisfiable by an assignment ψ.
With respect to ψ:

- At least one literal is assigned to 1 in every clause (?)
- Select a 1-literal in every tuple;
  These literals can be joined by edges (?)
  Yielding a k-clique in G. ♣

♣

$\langle G, k \rangle \in$ CLIQUE $\Longrightarrow$ $\phi \in$ 3SAT



Proof: Suppose G has a k-clique.

- No two of the cliques nodes are in the same triple. (?)
- G has k vertexes and k clauses, so each triple has exactly one clique node.
- Assign 1 to each node in clique;
  Assignment has no contradictions (?)
  Yielding a satisfying assignment to ϕ.

Recap

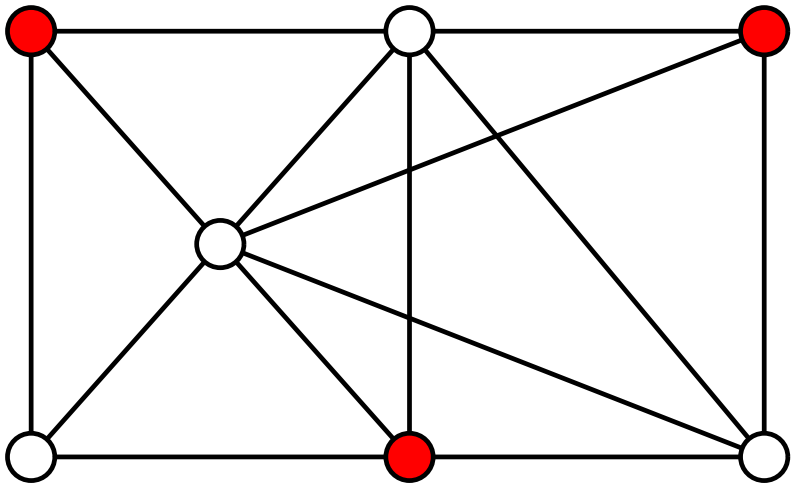
- We've constructed a poly-time computable function f.
- We saw that f has the property that $\phi \in$ 3SAT iff $f(\phi) \in$ CLIQUE.
- Therefore, f is a poly-time reduction from 3SAT to CLIQUE

  $\Longrightarrow$ 3SAT $\leq_P$ CLIQUE. ♣

Section 7

Independent Set

Independent Set



Definition 55

An independent set in a graph is a set of vertexes, no two of which are linked by an edge.

The language of independent sets:

$$\text{IND-SET} = \{\langle G, k\rangle : G \text{ contains an independent set of size } k\}$$

Clearly IND-SET $\in \mathcal{NP}$. We show that CLIQUE $\leq_P$ IND-SET and deduce that IND-SET $\in \mathcal{NPC}$

CLIQUE $\leq_P$ IND-SET

Proof:

Definition 56

The complement of a graph $G = (V, E)$ is a graph $G^c = (V, E^c)$, where $E^c = \{(v_1, v_2) : v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$.

Claim 57

If U is an independent set in G, then U is a clique in G^c.

The reduction from CLIQUE to IND-SET simply compute the complement of the graph. ♣

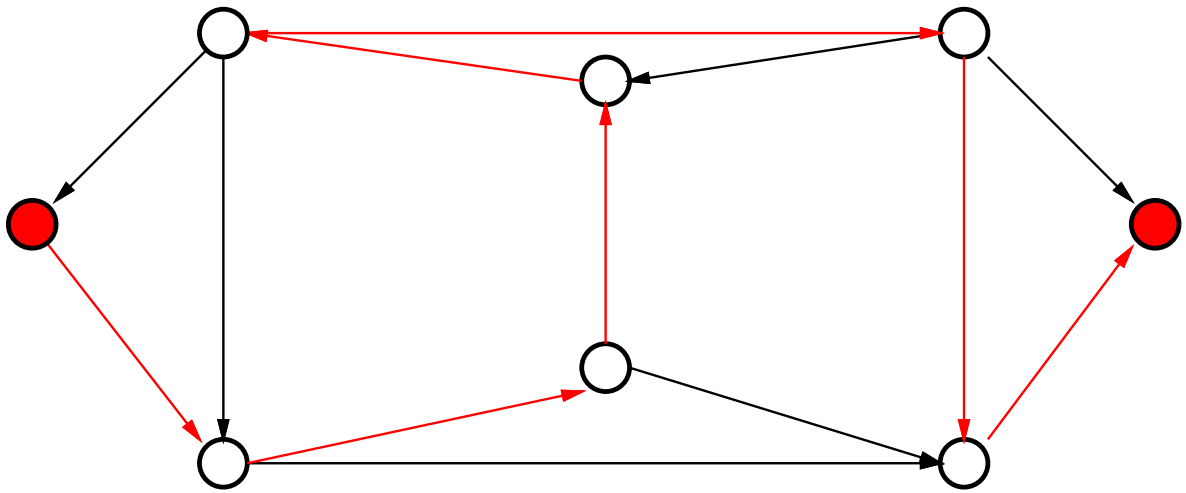
Remark 58

Same reduction shows that IND-SET $\leq_P$ CLIQUE

Section 8

Hamiltonian Paths and Cycles

Reminder – Hamiltonian Path



A Hamiltonian path in a directed G visits each node exactly once.

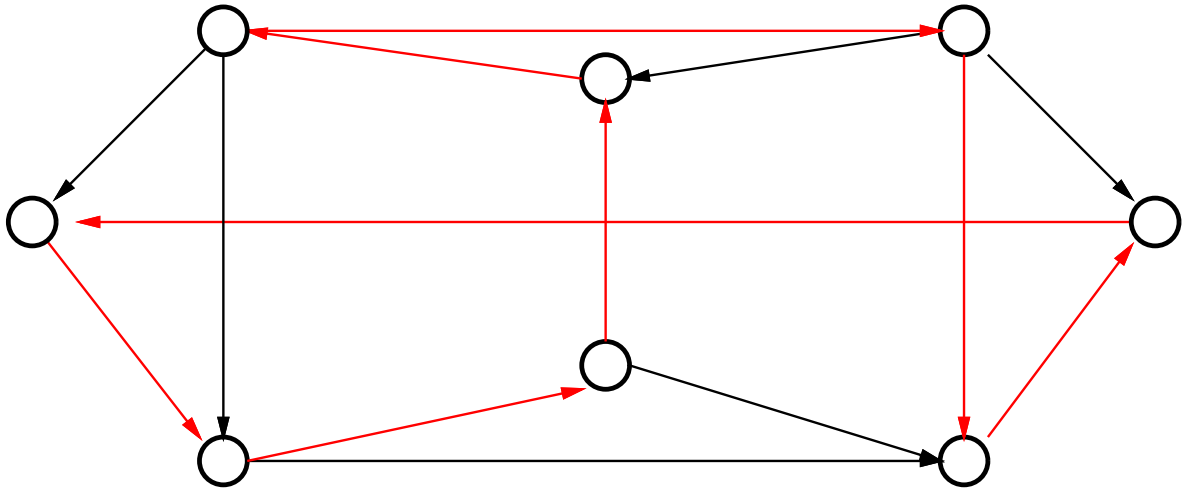
$$\text{HAMPATH} = \{\langle G, s, t\rangle : G \text{ has Hamiltonian path from } s \text{ to } t\}$$

Theorem 59

HAMPATH $\in \mathcal{NPC}$.

Not today . . .

Hamiltonian Cycle



A Hamiltonian Cycle in a graph is an Hamiltonian path that ends up where it starts (i.e., $s = t$).

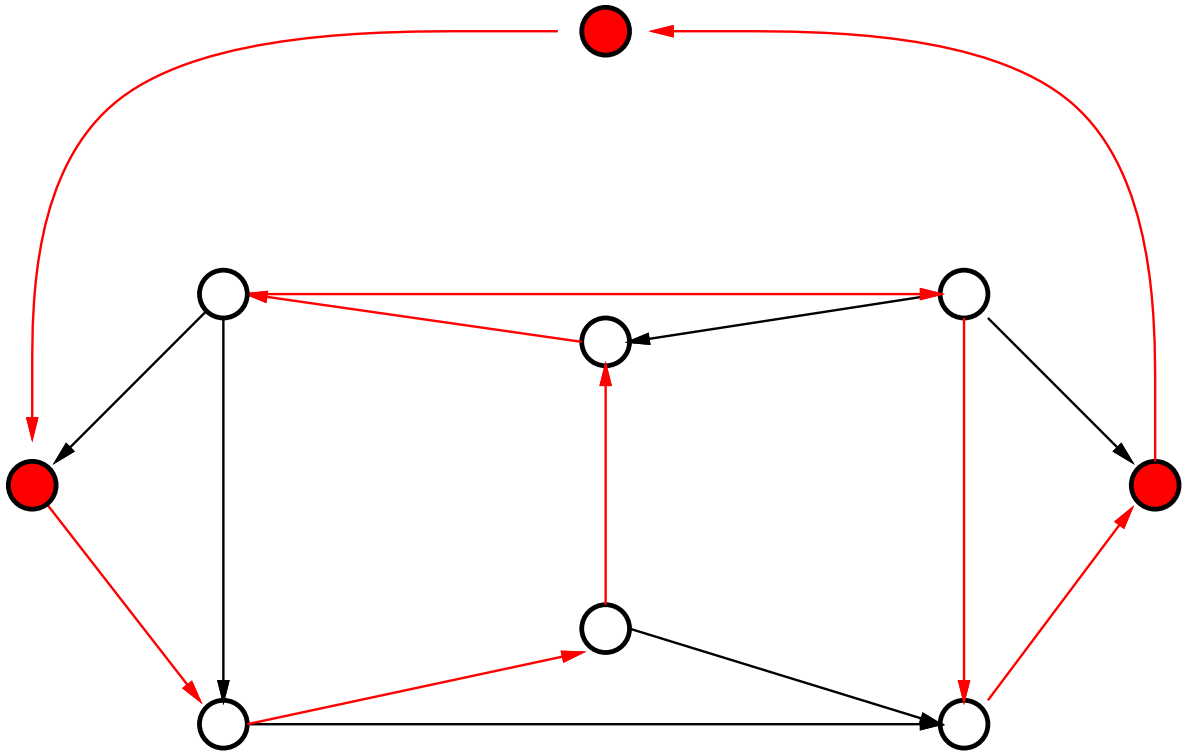
$$\text{HAMCYCLE} = \{\langle G \rangle : G \text{ has Hamiltonian cycle}\}$$

Clearly HAMCYCLE $\in \mathcal{NP}$. We will show that HAMPATH $\leq_P$ HAMCYCLE,

and deduce that HAMCYCLE $\in \mathcal{NPC}$

HAMPATH $\leq_P$ HAMCYCLE.

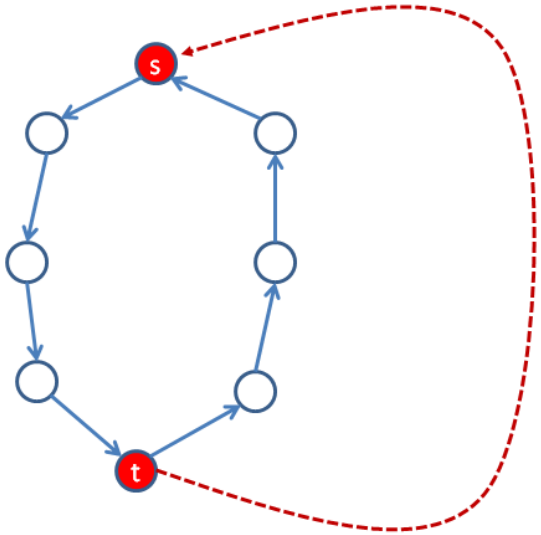
Proof's idea:



Hey, is the new vertex really needed? Why not just add an edge from t to s?

HAMPATH $\leq_P$ HAMCYCLE.

Why the new vertex really needed? Why not just add an edge from t to s?



HAMCYCLE $\leq_P$ HAMPATH.

Claim 60

HAMCYCLE $\leq_P$ HAMPATH.

Left as an easy (recommended) exercise.

Undirected Hamiltonian Circuit

$$\text{UHAMCYCLE} = \{\langle G \rangle : \ G \text{ is undirected } \& \text{ has Hamiltonian cycle}\}$$

where Hamiltonian cycle/path in an undirect graph, is defined analogously to the direct case.

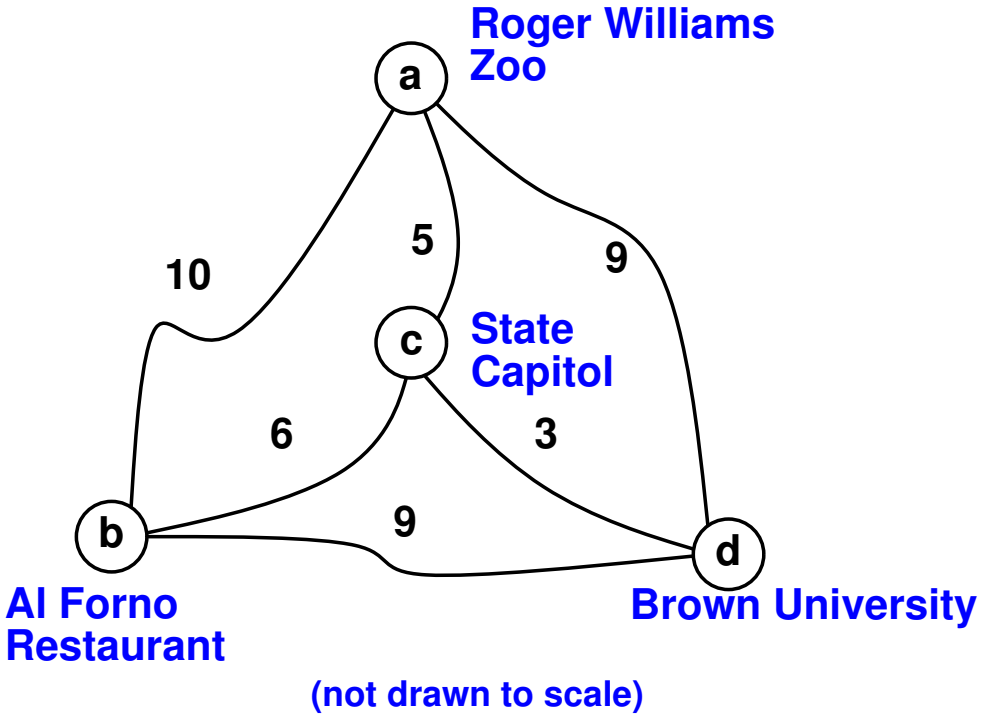
Clearly $\text{UHAMCYCLE} \in \mathcal{NP}$.

It is not hard to show (see Sipser 7.55, for a similar proof) that $\text{HAMCYCLE} \leq_P \text{UHAMCYCLE}$, and deduce that $\text{UHAMCYCLE} \in \mathcal{NPC}$

Section 9

Traveling Salesman

Traveling Salesman



Input: set of cities C and set of inter-city distances D

Goal: find a an hamiltonian cycle (TS tour) of total distance at most k

TRAVELING-SALESMAN $=$
$\{\langle C, D, k\rangle : (C, D)$ has a TS tour of total distance $\leq k\}$

UHAMCYCLE $\leq_P$ TRAVELING-SALESMAN

Proof: Given undirected graph $G = (V, E)$, construct traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from v_1 to v_2 is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.
- **correctness:** $\Longrightarrow$ Suppose G has a Hamiltonian cycle.
  - The distance assigned by reduction to all edges in this cycle is 1.
  - Thus in (C, D) there is a traveling salesman tour of total distance $|V| = k$ (namely $(C, D, k) \in$ TRAVELING-SALESMAN).

  $\Longleftarrow$ Suppose (C, D) has a traveling salesman tour of total distance $|V| = k$.
  - Tour cannot contain any edge of distance 2.
  - Therefore it gives a Hamiltonian cycle in G.
- **Efficiency:** reduction done in quadratic time (filling up distances for all edges of the complete graph).

Chains of Reductions: NPC Problems